(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 534 419 A2**

(12) # EUROPEAN PATENT APPLICATION

(21) Application number: 92116307.7

(51) Int. Cl.^{5}: **H04L 9/08**

(22) Date of filing: 11.09.92

(30) Priority: 27.09.91 US 766260

(43) Date of publication of application:
31.03.93 Bulletin 93/13

(84) Designated Contracting States:
AT CH DE DK ES FR GB IT LI NL SE

(71) Applicant: **International Business Machines Corporation**
Old Orchard Road
Armonk, N.Y. 10504(US)

(72) Inventor: **Matyas, Stephen M.**
10 298 Cedar Ridge Drive
Manassas, VA 22 110(US)
Inventor: **Johnson, Donald B.**
11 635 Crystal Creek Lane
Manassas, VA 22 111(US)
Inventor: **Le, An V.**
10 227 Battlefield Drive
Manassas, VA 22 110(US)
Inventor: **Prymak, Rostislaw**
15 900 Fairway Drive
Dumfries, VA 22 026(US)
Inventor: **Martin, William C.**
1835 Hilliard Lane
Concord, NC 28 025(US)
Inventor: **Rohland, William S.**
4234 Rotunda Road
Charlotte, NC 28 226(US)
Inventor: **Wilkins, John D.**
P.O. Box 8
Somerville, VA 22 739(US)

(74) Representative: **Herzog, Friedrich Joachim, Dipl.-Ing.**
IBM Deutschland Informationssysteme GmbH Patentwesen und Urheberrecht
Pascalstrasse 100
W-7000 Stuttgart 80 (DE)

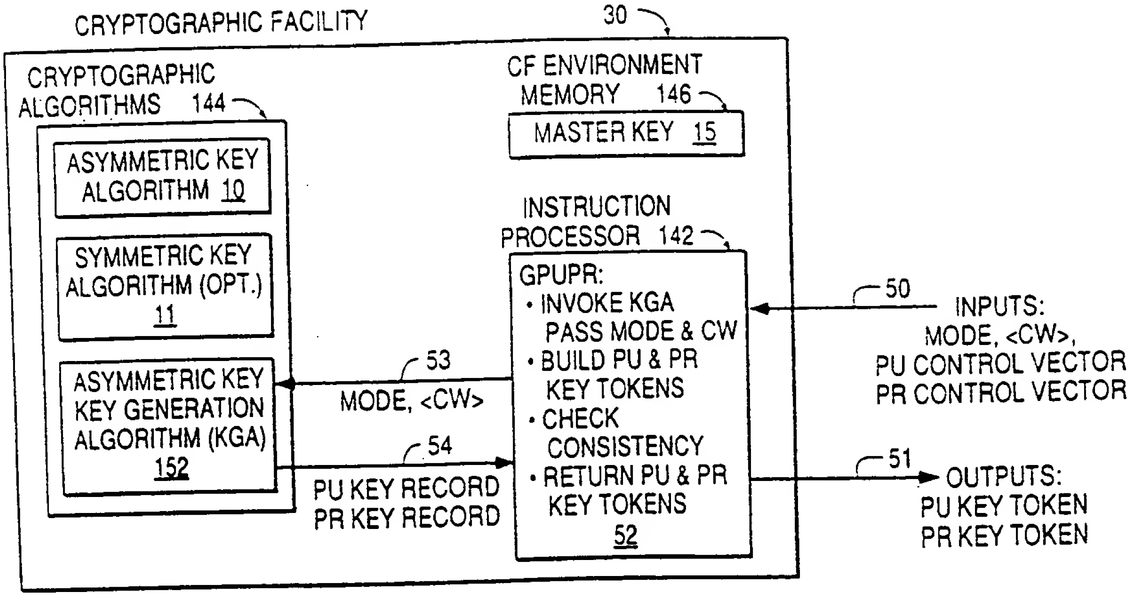
(54) **Public key cryptosystem key management based on control vectors.**

(57) A data processing system, method and program are disclosed, for managing a public key cryptographic system. The method includes the steps of generating a first public key and a first private key as a first pair in the data processing system, for use with a first public key algorithm and further generating a second public key and a second private key as a second pair in the data processing system, for use with a second public key algorithm. The method then continues by assigning a private control vector for the first private key and the second private key in the data processing system, for defining permitted uses for the first and second private keys. Then the method continues by forming a private key record which includes the first private key and the second private key in the data processing system, and encrypting the private key record under a first master key expression which is a function of the private control vector. The method then forms a private key token which includes the private control vector and the private key record, and stores the private key token in the data processing system.

At a later time, the method receives a first key use request in the data processing system, requiring the first public key algorithm. In response to this, the method continues by accessing the private key token in the data processing system and checking the private control vector to determine if the private key record contains a key having permitted uses which will satisfy the first request. The method then decrypts the private key record under the first master key expression in the data processing system and extracts the first private key from the private key record. The method selects the first public key algorithm in the data processing system for the first key use request and executes the first public key algorithm in the data processing system using the first private key to perform a cryptographic operation to satisfy the first key use request.

Rank Xerox (UK) Business Services
(3.10/3.5x/3.0.1)

FIG. 10


CRYPTOGRAPHIC FACILITY
30
CRYPTOGRAPHIC ALGORITHMS 144
ASYMMETRIC KEY ALGORITHM 10
SYMMETRIC KEY ALGORITHM (OPT.) 11
ASYMMETRIC KEY KEY GENERATION ALGORITHM (KGA) 152
CF ENVIRONMENT MEMORY 146
MASTER KEY 15
INSTRUCTION PROCESSOR 142
GPUPR:
• INVOKE KGA PASS MODE & CW
• BUILD PU & PR KEY TOKENS
• CHECK CONSISTENCY
• RETURN PU & PR KEY TOKENS
52
53
MODE, <CW>
54
PU KEY RECORD
PR KEY RECORD
50
INPUTS:
MODE, <CW>,
PU CONTROL VECTOR
PR CONTROL VECTOR
51
OUTPUTS:
PU KEY TOKEN
PR KEY TOKEN

BACKGROUND OF THE INVENTION

The invention disclosed broadly relates to data processing systems and methods and more particularly relates to cryptographic systems and methods for use in data processing systems to enhance security.

The following co-pending patent applications are related to this invention and are incorporated herein by reference:

B. Brachtl, et al., "Controlled Use of Cryptographic Keys Via Generating Stations Established Control Values," USP 4,850,017, issued July 18, 1989, assigned to IBM Corporation and incorporated herein by reference.

S. M. Matyas, et al., "Secure Management of Keys Using Control Vectors," USP 4,941,176, issued July 10, 1990, assigned to IBM Corporation and incorporated herein by reference.

S. M. Matyas, et al., "Data Cryptography Operations Using Control Vectors," USP 4,918,728, issued April 17, 1990, assigned to IBM Corporation and incorporated herein by reference.

S. M. Matyas, et al., "Personal Identification Number Processing Using Control Vectors," USP 4,924,514, issued May 8, 1990, assigned to IBM Corporation and incorporated herein by reference.

S. M. Matyas, et al., "Secure Management of Keys Using Extended Control Vectors," USP 4,924,515, issued May 8, 1990, assigned to IBM Corporation and incorporated herein by reference.

S. M. Matyas, et al., "Secure Management of Keys Using Control Vectors with Multi-Path Checking," US serial number 07/596,637, filed October 12, 1990, assigned to IBM Corporation and incorporated here by reference.

S. M. Matyas, et al., "Secure Cryptographic Operations Using Alternate Modes of Control Vector Enforcement," US serial number 07/574,012, filed August 22, 1990, assigned to IBM Corporation and incorporated here by reference.

S. M. Matyas, et al., "Secure Key Management Using Programmable Control Vector Checking," USP 5,007,089, issued April 9, 1991, assigned to IBM Corporation and incorporated herein by reference.

S. M. Matyas, et al., "Secure Key Management Using Control Vector Translation," USP 4,993,069 issued Feb. 12, 1991, assigned to IBM Corporation and incorporated herein by reference.

B. Brachtl, et al., "Data Authentication Using Modification Detection Codes Based on a Public One Way Encryption Function," USP 4,908,861, issued March 13, 1990, assigned to IBM Corporation and incorporated herein by reference.

D. Abraham, et al., "Smart Card Having External Programming Capability and Method of Making Same," US serial number 004,501, filed January 19, 1987, assigned to IBM Corporation and incorporated herein by reference.

S. M. Matyas, "Technique for Reducing RSA Crypto Variable Storage", USP 4,736,423, issued Apr. 5, 1988, assigned to IBM Corporation and incorporated herein by reference.

S. M. Matyas, et al., "Method and Apparatus for Controlling the Use of a Public Key, Based on the Level of Import Integrity for the Key," US serial number 07/602,989, filed October 24, 1990, assigned to IBM Corporation and incorporated herein by reference.

S. M. Matyas, et al., "A Hybrid Public Key Algorithm/Data Encryption Algorithm Key Distribution Method Based on Control Vectors," US serial number 07/748,407, filed August 22, 1991, assigned to IBM Corporation and incorporated herein by reference.

S. M. Matyas et al., "Generating Public and Private Key Pairs Using a Passphrase," filed on the same day as the instant application, US serial number 07/766,533, filed actually September 27, 1991, assigned to IBM Corporation and incorporated herein by reference.

The cryptographic architecture described in the cited patents by S. M. Matyas, et al. is based on associating with a cryptographic key, a control vector which provides the authorization for the uses of the key intended by the originator of the key. The cryptographic architecture described in the cited patents by S. M. Matyas, et al. is based on the Data Encryption Algorithm (DEA), see American National Standard X3.92-1981, Data Encryption Algorithm, American National Standards Institute, New York (December 31, 1981), whereas the present invention is based on both a secret key algorithm, such as the DEA, and a public key algorithm. Various key management functions, data cryptography functions, and other data processing functions are possible using control vectors, in accordance with the invention. A system administrator can exercise flexibility in the implementation of his security policy by selecting appropriate control vectors in accordance with the invention. A cryptographic facility (CF) in the cryptographic architecture is described in the above cited patents by S. M. Matyas, et al. The CF is an instruction processor for a set of cryptographic instructions, implementing encryption methods and key generation methods. A memory in the cryptographic facility stores a set of internal cryptographic variables. Each cryptographic instruction is described in terms of a sequence of processing steps required to transform a set of input parameters to a set of output parameters. A cryptographic facility application program (CFAP) is also described in the referenced patents and patent applications, which defines an invocation method, as a calling sequence, for each cryptographic instruction consisting of an instruction mnemonic and an address with corresponding input and output parameters.

Public key encryption algorithms are described in a paper by W. Diffie and M. E. Hellman entitled "Privacy and Authentication: An Introduction to Cryptography," Proceedings of the IEEE, Volume 67, No. 3, March 1979, pp. 397-427. Public key systems are based on dispensing with the secret key distribution channel, as long as the channel has a sufficient level of integrity. In a public key cryptographic system, two keys are used, one for enciphering and one for deciphering. Public key algorithm systems are designed so that it is easy to generate a random pair of inverse keys PU for enciphering and PR for deciphering and it is easy to operate with PU and PR, but is computationally infeasible to compute PR from PU. Each user generates a pair of inverse transforms, PU and PR. He keeps the deciphering transformation PR secret, and makes the enciphering transformation PU public by placing it in a public directory. Anyone can now encrypt messages and send them to the user, but no one else can decipher messages intended for him. It is possible, and often desirable, to encipher with PU and decipher with PR. For this reason, PU is usually called a public key and PR is usually called a private key. A corollary feature of public key cryptographic systems is the provision of a digital signature which uniquely identifies the sender of a message. If user A wishes to send a signed message M to user B, he operates on it with his private key PR to produce the signed message S. PR was used as A's deciphering key when privacy was desired, but it is now used as his "enciphering" key. When user B receives the message S, he can recover the message M by operating on the ciphertext S with A's public PU. By successfully decrypting A's message, the receiver B has conclusive proof it came from the sender A. Examples of public key cryptography are provided in the following U. S. patents: USP 4,218,582 to Hellman, et al., "Public Key Cryptographic Apparatus and Method;" USP 4,200,770 to Hellman, et al., "Cryptographic Apparatus and Method;" and USP 4,405,829 to Rivest, et al., "Cryptographic Communications System and Method."

Most cryptographic systems make use of many different types of keys, so that information encrypted with a key of one type is not affected by using a key of another type. A key is assigned a type on the basis of the information the key encrypts or the use being made of the key. For example, a data- encrypting key encrypts data. A key-encrypting key encrypts keys. A PIN-encrypting key encrypts personal identification numbers (PINs) used in electronic funds transfer and point-of-sale applications. A MAC key is used to generate and authenticate message authentication codes (MACs).

The use of encryption is based on a strategy of protecting a large amount of information (a data file or communications session) with a smaller additional amount of information (a single key). Sophisticated key hierarchies have been devised using this principle. For example, US Patents 4,850,017, 4,941,176, 4,918,728, 4,924,514, which are based on a symmetric key algorithm such as the Data Encryption Algorithm (DEA), make use of a key hierarchy wherein keys belonging to a cryptographic device are encrypted with a single master key and stored in a key data set. The master key is stored in clear form within the cryptographic hardware. The concept of using a single master key to encrypt keys stored in a key data set is known as the master key concept (see C.H. Meyer and S.M. Matyas, Cryptography--A New Dimension in Computer Data Security, John Wiley & Sons, Inc., New York, 1982.). Until now, the master key concept has been applied only to cryptographic systems based on a symmetric key cryptographic algorithm. However, the present invention extends the master key concept and teaches how it may be applied to cryptographic systems based on an asymmetric key cryptographic algorithm, and more particularly how it may be applied to cryptographic systems incorporating both asymmetric and symmetric key cryptographic algorithms, generally called hybrid cryptographic systems. The reader will appreciate that in a public key based cryptographic system employing (1) an asymmetric algorithm or (2) both asymmetric and symmetric algorithms, there is still a need to use many public and private keys pairs. Hence, at a minimum, the private keys must be stored in encrypted form outside the cryptographic hardware.

In order for a cryptographic system employing the master key concept to be made operable, each device must first be initialized with a master key and one or more other keys to permit the cryptographic system to communicate cryptographically with other cryptographic systems or to distribute keys to other cryptographic systems. Typically, these keys are generated and installed using manual entry techniques. In a well-designed cryptographic system, all other keys are generated and handled by the cryptographic system automatically. Keys generated by the cryptographic system are stored in encrypted form in a cryptographic key data set or transmitted in encrypted form to a designated receiving device where the key is imported (i.e., reencrypted to a form suitable for storage and use at the receiving device). Thus, an important feature of any key management scheme is the method used to encrypt keys for safe storage in a cryptographic key data set.

At the time a key is generated, the user or user application determines, from among the range of options permitted by the key management, the form of each generated key. For example, a generated key can be produced (1) in clear form, (2) in encrypted form suitable for storage in a cryptographic key data set, or (3) in encrypted form suitable for distribution to a designated receiving device. Generally, cryptographic systems have different options for generating keys in these different forms. Also, at the time a key is generated, the user or user application determines, from among the range of options permitted by the key management, the type and usage of each generated key. Type and usage information are examples of a class of key-related information called control information. For example, in US Patents 4,850,017, 4,941,176, 4,918,728, 4,924,514, 4,924,515, and 5,007,089, and IBM dockets MA9-88-033 (EP application no. 90105905.5) and BT9-90-018 (EP application no. 91109953.9) the control information is embodied within a data variable called the control vector. The control vector concepts taught in these US Patents and IBM dockets is summarized in a paper by S. M. Matyas entitled "Key handling with control vectors," IBM Systems Journal, Volume 30, No. 2, 1991, pp. 151-174.

In a cryptographic system employing control vectors, every key K has an associated control vector C. Thus, K and C denote a 2-tuple, where K initializes the cryptographic algorithm by selecting an enciphering transformation and C initializes the cryptographic hardware by selecting a set of cryptographic instructions, modes, and usage that K is granted. Implementation of the control vector concept requires that K and C be coupled cryptographically. Otherwise, the key-usage attributes granted to K by C could be changed by merely replacing C with another control vector. The method for accomplishing this is based on integrating C into the functions used to encrypt and decrypt keys, called control vector encryption (CVE) and control vector decryption (CVD). Fig. 1 is a block diagram illustration showing the implementation of the CVE and CVD algorithms within a cryptographic facility 30. CF 30 contains a CVE algorithm 1, a CVD algorithm 2, a master key (KM) 3, a to-be-encrypted key K 4, and a recovered key K 5. The CVE algorithm 1 encrypts a clear key K 4 within CF 30 using a variant key KM + C formed as the Exclusive OR product of master key KM 3 stored within CF 30 and control vector C 6 specified as an input to CF 30 to produce an output encrypted key value of the form e*KM + C(K) 7. Note that "+" denotes the Exclusive OR operation and e* denotes encryption with a 128-bit key. The operation of encryption consists of encrypting K with the leftmost 64 bits of KM + C then decrypting the result with the rightmost 64 bits of KM + C and then encrypting that result with the leftmost 64 bits of KM + C. The CVD algorithm 2 decrypts the encrypted key e*KM + C(K) 9 specified as an input to CF 30 with the variant key KM + C formed as the Exclusive-OR product of master key KM 3 stored within CF 30 and control vector C 8 specified as an input to CF 30 to produce an output clear key K 5. The operation of decryption consists of decrypting e*KM + C(K) with the leftmost 64 bit of KM + C then encrypting the result with the rightmost 64 bits of KM + C and then decrypting that result with the leftmost 64 bits of KM + C. The CVE algorithm is used to encrypt and protect keys stored outside the CF. The CVD algorithm is used to decrypt and recover keys to be processed within the CF.

Fig. 2 is a block diagram illustration of the control vector encryption (CVE) algorithm. Referring to Fig. 2, C is an input control vector whose length is a multiple of 64 bits; KK is a 128-bit key-encrypting key consisting of a leftmost 64-bit part KKL and a rightmost 64-bit part KKR, i.e., KK = (KKL,KKR); K is a 64-bit key or the leftmost or rightmost 64-bit part of a 128-bit key to be encrypted. The specification of KK is meant to be very general. For example, KK can be the master key KM, or some other key-encrypting key. The inputs are processed as follows. Control vector C is operated on by hashing algorithm ha, described below, to produce the 128-bit output hash vector H. H is Exclusive-ORed with KK to produce 128-bit output KK + H. Finally, K is encrypted with KK + H to produce output e*KK + H(K), where e* indicates encryption with 128-bit key KK + H using an encryption-decryption- encryption (e-d-e) algorithm as defined in ANSI Standard X9.17-1985 entitled "American National Standard for Financial Institution Key Management (Whole- sale)", 1985, and in ISO Standard 8732 entitled "Banking--Key Management (Wholesale)", 1988.

Fig. 3 is a block diagram illustration of the control vector decryption (CVD) algorithm. Referring to Fig. 3, C is an input control vector whose length is a multiple of 64 bits; KK is a 128-bit key-encrypting key consisting of a leftmost 64-bit part KKL and a rightmost 64-bit part KKR, i.e., KK = (KKL,KKR); e*KK + H(K) is the encrypted key to be decrypted. Control vector C is operated on by hashing algorithm ha, described below, to produce the 128-bit output hash vector H. H is Exclusive-ORed with KK to produce 128-bit output KK + H. Finally, e*KK + H(K) is decrypted with KK + H using a decryption- encryption-decryption (d-e-d) algorithm to produce output K. The d-e-d algorithm is just the inverse of the e-d-e algorithm.

Fig. 4 is a block diagram illustration of hashing algorithm ha. Hashing algorithm ha operates on input control vector C (whose length is a multiple of 64 bits) to produce a 128-bit output H, where H = ha(C). If C is 64 bits, ha(C) is set equal to (C,C), where the comma denotes concatenation, and the extension field (bits 45,46) in ha(C) is set equal to B'00'. That is, ha acts like a concatenation function. If C is 128 bits, ha(C) is set equal to C, and the extension field in ha(C) is set equal to B'01'. That is, ha acts like an identity function. If C is greater than 128 bits, ha(C) is set equal to a 128-bit one way cryptographic function of C, e.g. a 128-bit modification detection code calculated by the MDC-2 algorithm in Fig. 5, and the extension field in ha(C) is set equal to B'10'. In each of the three cases, the eighth bit of each byte in ha(C) is adjusted such that each byte has even parity. This adjustment ensures that when ha(C) is exclusive-ORed with KK, the variant key KK+h(C) has the same parity as KK. The extension field in ha(C) serves to ensure, for a fixed KK, that the set of keys of the form KK+h(C) consists of three disjoint subsets S1, S2, and S3, where S1 denotes the keys resulting from all 64-bit control vectors, S2 denotes the keys resulting from all 128-bit control vectors, and S3 denotes the keys resulting from all control vectors larger than 128 bits. This prevents a form of cheating wherein the CVD algorithm is tricked into decrypting an encrypted key using a false control vector. Hashing algorithm ha fulfills two important objectives. First, it handles both short and long control vectors, thus ensuring that a key-management scheme based on the control vector concept is openended. Second, the processing overhead to handle short control vectors (64 and 128 bits) is minimized so as to have minimal impact on the key management scheme.

As an alternate embodiment, the length of the input control vector to the hashing algorithm ha can be encoded in the extension field (bits 45,46). If the input control vector is 64 bits long, the field is B'00', if the input control vector is 128 bits long, the field is set to B'01' and if the input control vector is longer than 128 bits, the field is set to B'10'. This has the advantage of simplifying the hashing algorithm ha so that it does not need to set the extension field in the resulting output H, except if the input control vector was greater than 128 bits.

Fig. 5 is a block diagram illustration of a cryptographic function for calculating a 128-bit modification detection code (MDC), called the MDC-2 algorithm. Referring to Fig. 5, K1=X'5252525252525252' and L1=X'2525252525252525' are two 64-bit nonsecret constant keys. They are used only to process the first 64-bit block of plaintext, Y1. Thereafter, input value K2, K3, ... etc. are based on output values (A1,D1), (A2,D2), ..., etc., and input values L2, L3, ..., etc. are based on output values (C1,B1), (C2,B2), ..., etc. That is, the outputs of each iteration are fed back and used as the keys at the next iteration. The 32-bit swapping function merely replaces 32-bit value B with 32-bit value D and 32-bit value D with 32-bit value B.

In summary, the prior art describes a method for controlling key usage in cryptographic systems based on a symmetric key cryptographic algorithm such as the DEA. Key usage information is stored in a control vector C which is cryptographically coupled with the key K using control vector encryption and control vector decryption algorithms, CVE and CVD, respectively. The CVE and CVD algorithms can handle both short and long control vectors. The only restriction on length is that the control vector must be a multiple of 64 bits. The control vector itself consists of a group of subfields, where each subfield has it own definition and use within the key management to control the processing of the key. Encoding the control vector as a group of independent subfields has many advantages. The processing control vector checking need only concern itself with those subfields that pertain to the requested key usage. Thus, while the control vector may have many subfields, a particular cryptographic instruction may only need to check the encoded information in a few subfields. This speeds up the control vector checking process. Another important characteristic of the control vector is that the control vector accompanies (either explicitly or implicitly) the key wherever it goes. This is because the correct non-secret control vector must be specified to recover the correct secret key value. Thus, the control vector is available and can be checked at many different places within the cryptographic system: application program, cryptographic software, and cryptographic hardware.

Within a cryptographic system, the CVE and CVD algorithms are implemented so that their operation is transparent to the system. All clear keys are encrypted with the CVE algorithm before the keys are output from the cryptographic hardware. All encrypted keys are decrypted with the CVD algorithm before they are processed within the cryptographic hardware. Even within the cryptographic hardware, these services can be provided transparently from the cryptographic instructions that process keys. By employing a single pair of control vector encryption and decryption functions, most of the complexity associated with key handling can be encoded as information fields within the control vector and within the checking processes themselves, whereas the process of encrypting and decrypting keys and linking control vector information to the key can be handled with one common method.

The present invention provides a method for incorporating control vectors into a key management scheme that uses a public key algorithm. The reader will appreciate that while the advantages of controlling key usage with the control vector are universal in nature, the methods for accomplishing this can vary depending on the attributes of the cryptographic algorithm employed. For example, consider the method of encrypting K with a variant key KK+C to produce eKK+C(K). In this case, K is encrypted using the Data Encryption Algorithm, in which case the Exclusive-OR product of KK and C is always guaranteed to produce a valid DEA key, as DEA keys, ignoring parity bits, are maximally dense in the set of all binary numbers of their magnitude. When the cryptographic algorithm is an asymmetric algorithm such as the RSA algorithm, there are two keys PU and PR. In general, if (PU,PR) is a valid key pair, then (PU + C,PR + C) is not a valid key pair for an arbitrary value C. This is because the PU and PR key values meet certain mathematical constraints and are sparse in the set of all binary numbers of their magnitude. Thus, an alternate method for coupling C to PU and PR is needed. Moreover, encrypting one key with another can sometimes be cumbersome, e.g., when an the RSA algorithm is employed it is cumbersome to encrypt a key of one modulus value with a key of another modulus value if the value of the first modulus is greater than the value of the second modulus. This cumbersome situation must be dealt with in the underlying design so that a general methodology is achieved. The present invention will show how this is accomplished. In hybrid cryptographic systems where both a symmetric and asymmetric algorithm are implemented, the public and private keys belonging to the asymmetric algorithm can be encrypted with keys belonging to the symmetric key algorithm. In that case, the method for coupling a key and control vector can be similar to that described in the prior art. However, even here there are subtle differences that affect the design choice. For example, the public and private keys belonging to the asymmetric key algorithm are typically longer than the keys belonging to the symmetric key algorithm. Also, the possibility that the public and private keys will be of different and varying lengths must be addressed. 512-bit RSA keys are not uncommon, where a DEA master key is generally 128 bits. Thus, the CVE and CVD algorithms must be adjusted to permit long asymmetric keys to be encrypted with shorter (e.g., 128-bit) symmetric keys. Another difference is that, in theory, the public keys need not be encrypted when stored in a cryptographic key data set. However, there are advantages to handling both the public and private keys similarly. As examples, the same method for coupling the control vector and the private key can be used to couple the control vector and the public key, and the same method of authenticating the key value can be used. Also, handling the public and private keys in the same way means that all keys are handled and processed just one way, which reduces the complexity of the key management design. That is, as the private key must be encrypted to ensure that its value does not become known, the public key may also be encrypted to simplify the internal key management design, as then the key (whether public or private) will always be decrypted before being processed further.

When a public key algorithm is employed, the key lengths or key sizes are not fixed by the algorithm as with the DEA. In this case, the cryptographic system will most likely have to operate with public and private keys of different lengths, varying as much as several hundred bits. Therefore, the CVE and CVD algorithms must be designed to handle public and private keys with varying lengths. It is also important that the length of the key be made transparent from the application and the cryptographic system using the key.

In cryptographic systems based on the DEA, many cryptographic instructions that handle bulk data must be streamlined so that performance is not degraded by the introduction of the control vector and the encryption and decryption algorithms (CVE and CVD). However, when a public key (PK) algorithm is employed, the individual steps of encryption and decryption are orders of magnitude slower than encryption and decryption with the DEA. Thus, the design of a key management scheme based on a PK algorithm can have different underlying objectives. For example, key processing and key handling operations that introduce unwarranted processing overhead in a DEA-based key management, may indeed be appropriate for a PK-based key management. This is because the processing overhead while large compared to one DEA encryption may be insignificant compared to one PK encryption. In the present invention, a strategy is pursued of authenticating a key dynamically within the cryptographic hardware as part of the CVD algorithm. Relatively speaking, while this introduces significant processing overhead in a DEA-based key management scheme, it adds very little processing overhead in a PK-based key management scheme. However, this ensures that valid and strong PR and PU keys are used, and that an invalid (i.e., insecure) key value is not inadvertently used.

OBJECTS OF THE INVENTION

It is therefore an object of the invention to provide an improved method for controlling the usage of public and private keys.

It is another object of the invention to permit large amounts of control information for the public and private keys.

It is another object of the invention to permit the application, the system software, and the system hardware to check and set portions of the control information.

It is another object of the invention to permit keys to be authenticated within the crypto hardware as part of the key recovery process, so that all keys are authenticated before they are used by the crypto hardware.

It is another object of the invention to permit an open-ended design allowing new and expanded key usage to be added to the architecture.

It is another object of the invention to provide a single consistent method for handling both public and private keys.

It is another object of the invention to allow the physical makeup of the keys to appear transparent.

It is another object of the invention to allow users to port their public and private keys from one cryptographic system to another.

It is another object of the invention to base control vector encrypt and decrypt on a DEA master key of 128 bits.

It is another object of the invention to provide a general method for control vector encrypt and decrypt where the system master key is a private and public key pair of a commutative asymmetric cryptographic algorithm (i.e., no DEA or other symmetric algorithm master key is used).

It is another object of the invention to provide a general method for control vector encrypt and decrypt where the system master key is a quadruple of two key pairs of private and public keys of a non-commutative asymmetric cryptographic algorithm. Specifically the system master key quadruple consists of (1) a PU1 master key used to encrypt the public and private keys kept outside the cryptographic facility, (2) a PR1 master key used to decrypt the public and private keys kept outside the cryptographic facility, (3) a PR2 master key used to generate an authentication signature for the public and private keys kept outside the cryptographic facility, and (4) a PU2 master key used to verify the authentication signature of the public and private keys kept outside the cryptographic facility.

It is another object of the invention to provide a general method for control vector encrypt and decrypt where the system master key is a quadruple of one key pair of private and public keys using public key algorithm 1 and another key pair of private and public keys using public key algorithm 2. Specifically the system master key quadruple consists of (1) a PU1 master key (based on public key algorithm 1) used to encrypt the public and private keys kept outside the cryptographic facility, (2) a PR1 master key (using public key algorithm 1) used to decrypt the public and private keys kept outside the cryptographic facility, (3) a PR2 master key (using public key algorithm 2) used to generate an authentication signature for the public and private keys kept outside the cryptographic facility, and (4) a PU2 master key (using public key algorithm 2) used to verify the authentication signature of the public and private keys kept outside the cryptographic facility.

SUMMARY OF THE INVENTION

These and other objects, features, and advantages are accomplished by the invention as claimed and disclosed herein.

The invention describes a method for encrypting the public and private keys of a cryptographic asymmetric key (public key) algorithm, when these keys are stored outside the secure boundary of the cryptographic facility (i.e., cryptographic hardware) and for decrypting these keys when they are processed or used within the secure boundary of the cryptographic facility. The so-produced encrypted keys may be kept in a cryptographic key data set belonging to the cryptographic system software or they may be managed by the cryptographic application programs that use the keys. The public and private keys are encrypted by a system master key stored in clear form within the secure boundary of the cryptographic facility. In situations where the cryptographic system implements a symmetric key algorithm in addition to the asymmetric key algorithm the system master key can be a symmetric key. For example, if the cryptographic system implements both DEA and RSA algorithms, then the RSA public and private keys are protected with a 128-bit DEA master key.

In situations where the cryptographic system implements a commutative asymmetric key algorithm (such as the RSA algorithm), the system master key consists of a special public and private key pair (PU0,PR0) stored in clear form within the cryptographic facility. A commutative asymmetric key algorithm is one where the operation of encryption followed by decryption is equal to the operation of decryption followed by encryption in that both result in the original plaintext. The master public key PU0 is used to encrypt and verify authenticity for public and private keys stored outside the cryptographic facility and the master private key PR0 is used to decrypt and generate authentication signatures on the public and private keys stored outside the cryptographic facility. In addition to providing a means to encrypt and decrypt the public and private keys stored outside the cryptographic facility, the invention also provides a means to cryptographically couple the control vector with the public and private keys and to authenticate the public and private keys using a special authenticator produced within the cryptographic facility.

In situations where the cryptographic system implements only a non-commutative asymmetric key algorithm. the system master key may consist of a special quadruple composed of a two public and private key pairs ((PU1,PR1),(PU2,PR2)) stored in clear form within the cryptographic facility. A non-commutative asymmetric key algorithm is one where encryption must always be done before decryption. Master public key PU1 is used to encrypt public and private keys stored outside the cryptographic facility and master private key PR1 is used to decrypt public and private keys stored outside the cryptographic facility. Master public key PU2 is used to verify the authenticity of and private keys stored outside the cryptographic facility and master private key PR2 is used to generate authentication signatures for the public and private keys stored outside the cryptographic facility.

In situations where the cryptographic system implements two different asymmetric algorithms, where one algorithm is used for key encryption/decryption and another (different) algorithm is used for authentication. the system master key consists of a special quadruple composed of a two public and private key pairs ((PU1,PR1),(PU2,PR2)) stored in clear form within the cryptographic facility. (PU1,PR1) comprise an asymmetric key pair from a first public key algorithm and (PU2,PR2) comprise an asymmetric key pair from a second public key algorithm, which is different from the first algorithm. Master public key PU1 is used to encrypt public and private keys stored outside the cryptographic facility and master private key PR1 is used to decrypt public and private keys stored outside the cryptographic facility. Master public key PU2 is used to verify the authenticity of public and private keys stored outside the cryptographic facility and master private key PR2 is used to generate authentication signatures for the public and private keys stored outside the cryptographic facility.

Note also, as an alternate embodiment, if the public key algorithm is not commutative, if both the public key and the private keys that are used as the master key pair are kept secret, then only one master key pair is needed. In this case, the (secret) public key is used to encrypt the authentication record and the private key is used to decrypt it. Normally this would represent a security exposure, but as the public key is secret and known only inside the cryptographic facility, there is no exposure. Care must be taken to ensure that the (secret) public key is never inadvertently exposed.

Fig. 6 illustrates a cryptographic facility 30 containing a commutative asymmetric algorithm master key. In this case, the public and private keys stored outside the cryptographic facility 30 are protected (i.e., encrypted for privacy and authenticated) with an asymmetric master key pair, designated (PU0,PR0). Outside the cryptographic facility 30, all public and private keys are stored in key tokens. Public keys are stored in public key tokens (PU key tokens) and private keys are stored in private key tokens (PR key tokens). The PU key tokens and PR key tokens are stored in a cryptographic key data set 32 managed by the cryptographic system software, or they may be managed by the cryptographic application programs themselves (not shown in Fig. 6).

Fig. 7 illustrates a cryptographic facility 30 containing an asymmetric key algorithm and a symmetric key algorithm. In this case, the public and private keys stored outside the cryptographic facility 30 are protected with a symmetric system master key, designated KM. If the symmetric key algorithm is the DEA, then KM is a 128-bit key, as described in the prior art. As in Fig. 6, the public and private keys are stored in PU key tokens and PR key tokens. The PU key tokens and PR key tokens are stored in a cryptographic key data set 32 managed by the cryptographic system software, or they may be managed by the cryptographic application programs themselves (not shown in Fig. 7).

The reader will appreciate from the full description of the invention, provided below that, except for the special functions that encrypt and decrypt the keys in the key tokens, the means for protecting keys based on any of the following methods:

(1) a symmetric system master key (KM),

(2) a commutative asymmetric system master key pair (PU0,PR0),

(3) a non-commutative asymmetric master key pair (PU0,PR0) where both the public and private key are kept secret.

(4) a non-commutative asymmetric master key quadruple ((PU1,PR1),(PU2,PR2)), or

(5) a master key quadruple ((PU1,PR1),(PU2,PR2)) when the first key pair uses one public key algorithm for key encryption/decryption and the second key pair uses another public key algorithm different from the first for authentication can be made transparent to the user of a cryptographic system. Thus, the cryptographic instructions that process and use the public and private keys and the cryptographic software and cryptographic application programs that handle the public and private key tokens are unaffected by the particular encryption and decryption means for storage and recovery of the public and private keys. This is so because the keys are treated as logical entities. Their physical characteristics such as length, format, component make up, etc., are kept transparent to the cryptographic system. This is partially accomplished through the use of special records called the public key record (PU key record)

and private key record (PR key record) which may have varying length, as the keys they contain may have varying length. All public and private keys generated within the cryptographic system are stored in these varying-length key records. As an alternate embodiment, the key records may be set to a fixed size that will contain the largest size public and private keys that will be generated and/or used on the system.

Fig. 8 illustrates the production of public and private keys using a public key key generation algorithm (KGA) 152. In response to a request to generate a (PU,PR) key pair, public key generation algorithm 152 causes a (PU,PR) key pair to be generated. The generated public key PU is stored in a PU key record and the generated private key PR is stored in a PR key record. The PU key record and PR key record are returned as outputs. In addition to returning the PU key record and PR key record, the public key generation algorithm 152 may also optionally return a PU_length parameter indicating the length of PU key record and a PR_length parameter indicating the length of PR key record. The optional length parameters may be useful in implementations where the lengths of PU key record and PR key record may vary.

Fig. 9 illustrates the formats of the PU key record and PR key record. The PU key record contains parse data that permits the public key to be recovered from the record. The parse data may be length and displacement data of fields in the record. The PU key record also contains control information that may be useful in describing the record type and type of key or keys stored within the record. The PU key record also permits one or more public keys to be stored as a single logical public key. This may be particularly useful in situations where a first public key algorithm is used for DEA key encryption/decryption purposes, e.g., to distribute DEA keys from one device to another, and a second public key algorithm is used for generating and verifying digital signatures. Thus, a first public key PU1 is used to encrypt DEA keys and a second public key PU2 is used verify digital signatures. In such situations, the cryptographic system is designed in such a way that the key processing operations will know from the context of the operations being perform whether the public key to be used is PU1 or PU2. The PR key record also contains parse data that permits the private key to be recovered from the record. The PR key record also contains control information that may be useful in describing the record type and type of key or keys stored within the record. The PR key record also permits one or more private keys to be stored as a single logical private key. Thus, a first private key PR1 is used to decrypt a DEA key encrypted by the first public key PU1, and a second private key PR2 is used to generate digital signatures for later verification by the second public key PU2. In such situations, the cryptographic system is designed in such a way that the key processing operations will know from the context of the operations being performed whether the private key to be used is PR1 or PR2. The PU and PR key records keep algorithm specific and key specific information transparent to the cryptographic system. Only the public key algorithm itself that processes the key records need be aware of the internal structure and makeup of these key records.

As an alternate embodiment, in certain situations, there may be advantages to maintaining the logical key records in two forms: the first containing both the private keys and public keys for the owner or creator of the keys and the second containing just the public keys for distribution to others. As before, if using the owner's logical key record containing both private and public keys, the correct key to use can be determined from context.

Fig. 10 illustrates the production of public and private key pairs using a Generate Public and Private Key Pair (GPUPR) instruction. The GPUPR instruction is described in detail in co-pending patent application by S. M. Matyas, et al. entitled "Generating Public and Private Key Pairs Using a Passphrase", as cited in the background art. Referring now to Fig. 10, the GPUPR instruction 52 is contained in an instruction processor 142 within the cryptographic facility (CF) 30. In practice, the CF 30 is implemented within secure hardware, so that keys and cryptographic variables stored within the CF 30 are protected, i.e., both the secrecy and integrity of these keys and cryptographic variables are protected. The CF 30 also contains a CF environment memory 146 for the storage of keys and cryptographic variables such as a master key 15. Fig. 10 does not specify whether the master key is (1) a symmetric master key KM, (2) an asymmetric commutative master key pair (PU0,PR0), (3) a non-commutative asymmetric master key pair (PU0,PR0) where both the public and private key are kept secret, (4) an asymmetric non- commutative master key quadruple ((PU1,PR1),(PU2,PR2)), or (5) an asymmetric two-PK-algorithm master key quadruple (-(PU1,PR1),(PU2,PR2)) where the first pair uses one public key algorithm and the second pair uses a different public key algorithm from the first. The CF 30 also contains cryptographic algorithms 144, which includes an asymmetric key algorithm 10, an optional symmetric key algorithm 11, and an asymmetric key key generation algorithm (KGA) 152. The inputs to the GPUPR instruction at 50 consist of a mode, an optional code word, PU control vector, and PR control vector. In response to a request to execute the GPUPR instruction at 50, the GPUPR instruction invokes the KGA 152, at 53, passing the mode and optional code word. The mode indicates to KGA 152 whether the to-be-generated public and private key pair (PU,PR) are generated from a code word (mode='PP') or not (mode='no_PP'). In response the KGA 152 produces a public and private key pair (PU,PR) which are formatted in a PU key record and PR key record. The PU key record and PR key record are returned to the GPUPR instruction at 54. In response, the GPUPR instruction builds a PU key token and a PR key token containing the encrypted PU key record and encrypted PR key record, respectively. Each key token contains a control vector and an authenticator, as further described below. The GPUPR instruction 52 also performs consistency checking on the mode and control vector supplied as inputs at 50, see also co-pending patent application by S. M. Matyas, et al. entitled "Generating Public and Private Key Pairs Using a Passphrase", cited in the background art, for a further discussion of this consistency checking. The so-produced PU key token and PR key token are returned as outputs at 51.

Fig. 11 illustrates the formats of the PU key token and PR key token. The PU key token consists of a header, a PU control vector, an encrypted PU key record, and a PU authenticator. As an alternate embodiment, the PU key token may consist of a header, a PU control vector, a plaintext PU key record, and a PU authenticator. The preferred embodiment has an encrypted PU key record in the PU key token as the PR key token must contain an encrypted PR key record (to maintain its secrecy) and doing both PU and PR key tokens in the same manner simplifies the processing. The PR key token consists of a header, a PR control vector, an encrypted PR key record, and a PR authenticator. The header in the PU key token consists of information (e.g., offsets or displacements to start of fields, offsets or displacements to end of fields, and/or lengths of fields) that enable the system to determine the start and end of each other field in the PU key token. The PU control vector consists of a PU key type, PU key usage data, PR key usage data (for history purposes), algorithm identifier, algorithm-specific data, key start date/time, key expiration data/time, device identifier, user identifier, key identifier, logical device identifier, and user-defined data. The fields of PU control vectors are presented in more detail under "Description of the Best Mode for Carrying Out the Invention." If the system master key is a symmetric key KM, then PU key record is encrypted with a variant key derived from KM, as explained below. If the system master key is an asymmetric key pair (PU0,PR0), then PU key record is encrypted with PU0, as explained below. The PU authenticator is a special authentication code produced at the time the PU key token is constructed. Later, when the PU key token is specified as a parameter input to a cryptographic instruction, the PU authenticator is used to validate the public key as part of key recovery, before the recovered PU is processed within the cryptographic instruction.

The header in the PR key token consists of information (e.g., offsets or displacements to start of fields, offsets or displacements to end of fields, and/or lengths of fields) that enable the system to determine the start and end of each other field in the PR key token. The PR control vector consists of a PR key type, PR key usage data, PU key usage data (for history purposes), algorithm identifier, algorithm-specific data, key start date/time, key expiration data/time, device identifier, user identifier, key identifier, logical device identifier, and user-defined data. The fields of PR control vectors are presented in more detail under "Description of the Best Mode for Carrying Out the Invention." If the system master key is a symmetric key KM, then PR key record is encrypted with a variant key derived from KM, as explained below. If the system master key is an asymmetric key pair (PU0,PR0), then the PR key record is encrypted with PU0, as explained below. The PR authenticator is a special authentication code produced at the time the PR key token is constructed. Later, when the PR key token is specified as a parameter input to a cryptographic instruction, the PR authenticator is used to validate the public key as part of key recovery, before the recovered PR is processed within the cryptographic instruction.

In co-pending patent application by S. M. Matyas, et al. entitled "Generating Public and Private Key Pairs Using a Passphrase", cited in the background art, the outputs of key generator algorithm 152 are the generated public and private keys, PU and PR. Actually, the outputs are a PU key record and a PR key record, containing the generated PU and PR, respectively, as defined here. Those skilled in the art will appreciate that the description of the GPUPR instruction and the key generation algorithm in copending patent application by S. M. Matyas, et al. entitled "Generating Public and Private Key Pairs Using a Passphrase", is for all intents and purposes the same as the description provided here, and that returning PU and PR as outputs from the key generation algorithm 152, instead of return PU and PR key records does not depart from the underlying invention.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other objects, features, and advantages of the invention will be more fully appreciated with reference to the accompanying figures.

Fig. 1 is a block diagram illustration of the process to encrypting keys and decrypting keys in a DEA-based cryptographic system using the control vector encrypt (CVE) and control vector decrypt (CVD) algorithms.

Fig. 2 is a block diagram illustration of the CVE algorithm implemented in a DEA-based cryptographic system.

Fig. 3 is a block diagram illustration of the CVD algorithm implemented in a DEA-based cryptographic system.

Fig. 4 is a block diagram illustration of the hashing algorithm ha implemented in the CVE and CVD algorithms of Figs. 1, 2, and 3.

Fig. 5 is a block diagram illustration of the MDC-2 algorithm.

Fig. 6 is a block diagram illustration of a first embodiment of the invention wherein the generated public and private keys stored outside the cryptographic facility are protected with a commutative asymmetric system master key pair (PU0,PR0).

Fig. 7 is a block diagram illustration of a second embodiment of the invention wherein the generated public and private keys stored outside the cryptographic facility are protected with a symmetric system master key KM.

Fig. 8 is a block diagram illustration of a public key key generation algorithm (KGA).

Fig. 9 illustrates the formats of the PU key record and PR key record.

Fig. 10 is a block diagram illustration of the GPUPR instruction.

Fig. 11 illustrates the formats of the PU key token and the PR key token.

Fig. 12 illustrates a communications network 10 including a plurality of data processors, each of which includes a cryptographic system.

Fig. 13 is a block diagram of a cryptographic system 22.

Fig. 14 is a block diagram of a cryptographic facility 30.

Fig. 15 is a block diagram illustration of the cryptographic algorithms 144 component of the cryptographic facility 30 containing the key record encrypt and key record decrypt algorithms.

Fig. 16 is a flow diagram of a first embodiment of key record encrypt algorithm 12.

Fig. 17 is a flow diagram of a first embodiment of key record decrypt algorithm 13.

Fig. 18 is a flow diagram of a second embodiment of key record encrypt algorithm 12.

Fig. 19 is a flow diagram of a second embodiment of key record decrypt algorithm 13.

Fig. 20 is a functional block diagram illustrating the recovery of two private keys and their use in two public key algorithms to fulfill two different cryptographic service requests.

Fig. 21 is a block diagram showing the production of an internal key token from a key record and the production of an external key token from a key record.

Fig. 22 is a block diagram showing the production of an internal key token from an internal key unit produced from a key record and the production of an external key token from an external key unit produced from a key record.

Fig. 23 lists the components of the Instruction Processor 142.

Fig. 24 shows the elements of the Configuration Table in the CF Environment Memory 146.

Fig. 25 shows the main elements of the Cryptographic Algorithms 144.

Fig. 26 is a block diagram illustration of the components of the CF Environment.

Fig. 27 shows the instructions controlled by the DEFINE, AUTH CONTROL, AUTH, and ENABLE fields in the Configuration Vector.

Fig. 28 is a block diagram illustration of the MDC Table.

Fig. 29 is a block diagram illustration of the Counter Table.

Fig. 30 illustrates the control vector hierarchy of PKCD keys.

Fig. 31 is a block diagram illustration of the fields in a control vector associated with a private authentication key.

Fig. 32 is a block diagram illustration of the fields in a control vector associated with a private certification key.

Fig. 33 is a block diagram illustration of the fields in a control vector associated with a private key management key.

Fig. 34 is a block diagram illustration of the fields in a control vector associated with a private user key.

Fig. 35 is a block diagram illustration of the fields in a control vector associated with a public authentication key.

Fig. 36 is a block diagram illustration of the fields in a control vector associated with a public certification key.

Fig. 37 is a block diagram illustration of the fields in a control vector associated with a public key management key.

Fig. 38 is a block diagram illustration of the fields in a control vector associated with a public user key.

Fig. 39 is a block diagram illustration of the fields in a hash vector.

DESCRIPTION OF MODES FOR CARRYING OUT THE INVENTION

Environment Description: Fig. 12 illustrates a network block diagram showing a communications network 10 to which is connected a plurality of data processors including data processor 20, data processor 20', and data processor 20". Also included in each data processor is a cryptographic system, as shown in Fig. 12. Data processor 20 includes cryptographic system 22, data processor 20' includes cryptographic system 22' and data processor 20" includes cryptographic system 22". Each data processor supports the processing of one or more applications which require access to cryptographic services such as for the encryption, decryption and authenticating of application data and the generation and installation of cryptographic keys. The cryptographic services are provided by a secure cryptographic facility in each cryptographic system. The network provides the means for the data processors to send and receive encrypted data and keys. Various protocols, that is, formats and procedural rules, govern the exchange of cryptographic quantities between communicating data processors in order to ensure the interoperability between them.

Fig. 13 illustrates the cryptographic system 22. In the cryptographic system 22, the cryptographic facility (CF) 30 has an input 37 from a physical interface. The cryptographic facility access program (CFAP) 34 is coupled to the cryptographic facility 30 by means of the interface 31. The cryptographic key data set (CKDS) 32 is connected to the cryptographic facility access program 34 by means of the interface 33. The application programs (APPL) 36 are connected to the cryptographic facility access program 34 by means of the interface 35.

A typical request for cryptographic service is initiated by APPL 36 via a function call to the CFAP 34 at the interface 35. The service request includes key and data parameters, as well as key identifiers which the CFAP 34 uses to access encrypted keys from the CKDS 32 at the interface 33. The CFAP 34 processes the service request by issuing one or more cryptographic access instructions to the CF 30 at the interface 31. The CF 30 may also have an optional physical interface 37 for direct entry of cryptographic variables into the CF 30. Each cryptographic access instruction invoked at the interface 31 has a set of input parameters processed by the CF 30 to produce a set of output parameters returned by the CF 30 to the CFAP 34. In turn, the CFAP 34 may return output parameters to the APPL 36. The CFAP 34 may also use the output parameters and input parameters to subsequently invoke instructions. If the output parameters contain encrypted keys, then the CFAP 34, in many cases, may store these encrypted keys in the CKDS 32.

Fig. 14 illustrates the cryptographic facility 30. The cryptographic facility 30 is maintained within a secure boundary 140. The cryptographic facility 30 includes the instruction processor 142 which is coupled to the cryptographic algorithms 144 which are embodied as executable code. The cryptographic facility environment memory 146 is coupled to the instruction processor 142. The physical interface can be coupled over line 37 to the CF environment memory 146, as shown in the figure. The instruction processor 142 is coupled to the cryptographic facility access program (CFAP) 34 by means of the interface at 31.

The instruction processor 142 is a functional element which executes cryptographic microinstructions invoked by the CFAP access instruction at the interface 31. For each access instruction, the interface 31 first defines an instruction mnemonic or operation code used to select particular microinstructions for execution. Secondly a set of input parameters is passed from the CFAP 34 to the CF 30. Thirdly, a set of output parameters is returned by the CF 30 to the CFAP 34. The instruction processor 142 executes the selected instruction by performing an instruction specific sequence of cryptographic processing steps embodied as microinstructions stored in cryptographic microinstruction memory 144. The control flow and subsequent output of the cryptographic processing steps depend on the values of the input parameters and the contents of the CF environment memory 146. The CF environment memory 146 consists of a set of cryptographic variables, for example keys, flags, counters, CF configuration information, etc., which are collectively stored within the CF 30. The CF environment variables in memory 146 are initialized via the interface 31, that is by execution of certain CF microinstructions which read input parameters and load them into the CF environment memory 146. Alternately, initialization can be done via an optional physical interface which permits cryptographic variables to be loaded directly into the CF environment memory 146, for example via an attached key entry device.

The physical embodiment of the cryptographic facility secure boundary 140, incorporates the following physical security features. The physical embodiment resists probing by an insider adversary who has limited access to the cryptographic facility 30. The term "limited" is measured in minutes or hours as opposed to days or weeks. The adversary is constrained to a probing attack at the customer's site using limited electronic devices as opposed to a laboratory attack launched at a site under the control of the adversary using sophisticated electronic and mechanical equipment. The physical embodiment also detects attempts at physical probing or intruding, through the use of a variety of electro-mechanical sensing devices. Also, the physical embodiment of the cryptographic facility 30 provides for the zeroization of all internally stored secret cryptographic variables. Such zeroization is done automatically whenever an attempted probing or intrusion has been detected. The physical embodiment also provides a manual facility for a zeroization of internally stored secret cryptographic variables. Reference to the Abraham, et al. patent application cited above, will give an example of how such physical security features can be implemented.

Key Record Encryption/Decryption: Fig 15 is a block diagram illustration of cryptographic facility 30 incorporating the key record encrypt and key record decrypt algorithms. Cryptographic facility 30 contains an instruction processor 142 consisting of a plurality of cryptographic instructions (not shown in Fig. 15), a CF environment memory 146 containing a master key 15, and cryptographic algorithms 144. Cryptographic algorithms 144 contains an asymmetric key cryptographic algorithm 10, an optional symmetric-key cryptographic algorithm 11, an asymmetric-key key generation algorithm 152, a key record encrypt algorithm 12, and a key record decrypt algorithm 13. Key record encrypt algorithm 12 is a low-level function used by instruction processor 142 to encrypt a key record (PU key record or PR key record) and produce an encrypted key authenticator record (KAR), which serves to authenticate the key record and associated control vector to the cryptographic facility 30. During key generation (via the GPUPR instruction), the PU and PR key records produced by the asymmetric key key generation algorithm 152 are encrypted and then stored in key tokens constructed by the instruction processor. These key tokens are returned as outputs at 51. The key record encrypt algorithm 12 is invoked by the instruction processor 142 at 14, passing a key record and control vector. In response, key record encrypt algorithm 12 encrypts the key record with master key 15, or a variant key derived from master key 15, as explained below. Key record encrypt algorithm 12 also produces a key authenticator record (KAR) from the key record or from the control vector and key record, again as explained below. The so-produced KAR is then encrypted with master key 15, or a variant key derived from master key 15 (different from the variant key used to encrypt the key record), as explained below. Note that if the KAR was not encrypted, this might represent a security exposure, as the control vector and key record for a public key and the KAR generation algorithm are all assumed to be public knowledge. This would possibly allow substitution of a incorrect public key or incorrect control vector for the correct values, for example, in the cryptographic key data set. While the KAR for a private key may not need to be encrypted for security, in the preferred embodiment, it is encrypted to allow consistent processing of the KAR for both public and private keys. As an alternate embodiment, the KAR for the private key could just be the output of a strong cryptographic one-way function, such as the MDC-2 function described elsewhere. The encrypted key record and encrypted KAR are returned at 16 to the instruction processor 142. Key record decrypt algorithm 13 is a low-level function used by instruction processor 142 to decrypt a key record (PU key record or PR key record) and authenticate the key record and associated control vector to the cryptographic facility 30 before permitting instruction processor 142 to process or use the key in the decrypted key record. Many of the cryptographic instructions executing in the instruction processor 142 make use of cryptographic keys stored in key tokens and supplied as inputs at 50 to the instruction processor 142. Before a key can be processed or used by the instruction processor 142, it must be recovered. During key recovery, the encrypted PU and PR key records contained in the input key tokens (at 50) are decrypted and authenticated. The key record decrypt algorithm 13 is invoked at 17 by the instruction processor 142, passing a key record and control vector as inputs. In response, key record decrypt algorithm 13 decrypts the encrypt key record with master key 15, or a variant key derived from master key 15, as explained below. Key record decrypt algorithm 13 also produces a key authenticator record (KAR) from the recovered key record, or from the control vector and recovered key record, again as explained below. The key record decrypt algorithm 13 then decrypts the encrypted KAR and compares the recovered value of KAR and the generated or produced KAR for equality. If the two values of KAR are equal, the key record decrypt algorithm 13 returns the recovered key record and a return code (e.g., RC=0) indicating that the key record has been successfully authenticated via the KAR. Otherwise, if the two value of KAR are unequal, the key record decrypt algorithm 13 returns only a return code (e.g., RC=1) indicating that the key record has failed to be authenticated via the KAR.

Fig. 16 is block diagram illustration of a first embodiment of the key record encrypt algorithm 12. The first embodiment of the invention covers the case where the cryptographic system implements both a symmetric key algorithm and an asymmetric key algorithm, and where the master key used to encrypt the key records in the key tokens stored outside the cryptographic facility is a symmetric key KM. Referring now to Fig. 16, the inputs (a) key record and (b) control vector are read at step 500. Key record is the key record to be encrypted and control vector is key-related data, or data related to the key stored in key record. Control vector is the same control vector stored in the key token, as described in Fig. 11. At step 501, a hash value HASH1 is calculated on the control vector using hash algorithm ha1. For example, when the master key is a 128-bit DEA master key, HASH1 can be a 128-bit MDC calculated with the MDC-2 algorithm of Fig. 5. At step 502, hash vectors H1 and H2 are calculated from HASH1. For example, when the master key is a 128-bit DEA master key and H1 and H2 are both 128-bit hash vectors, the procedure for calculating H1 and H2 is as follows. The 128-bit hash vector H1 is calculated from HASH1 as follows:

1. Set bit 30 of HASH1 equal to B'0'.
2. Set bit 38 of HASH1 equal to B'1'.
3. Set bits 45..46 of HASH1 equal to B'10'.
4. Set bit 62 of HASH1 equal B'0'.
5. For each byte in HASH1 (bits are numbered b0 through b7), set bit b7 so that bits b0 through b7 have an even number of one bits (i.e., to have even parity).

Bits 30 and 38 are anti-variant bits whose values are set so that the resulting hash vector H is guaranteed to be different from a variant value in which each byte of the variant has the same bit pattern. Bits 45 and 46 are set to B'10' to distinguish H1 from a 64-bit control vector (bits 45..46 equal to B'00') and a 128-bit control vector (bits 45..46 equal to B'01'). In this case, B'10' indicates that H1 has been derived from a "long" control vector whose length exceeds 128 bits. Bit 62 indicates whether the control vector is associated with a key record (B'0') or a key authenticator record (B'1'). The 128-bit hash vector H2 is calculated from H1 as follows:

1. Set H2 equal to H1.
2. Set bit 62 of H2 equal to B'1'.
3. Invert bit 63 of H2 (i.e., the parity bit).

Basically, H2 differs from H1 only in that H1 is associated with a key record (bit 62 equals B'0') and H2 is associated with a key authenticator record (bit 62 equals B'1'). The parity bit is adjusted to maintain even parity. Otherwise, H1 and H2 are equal. At step 503, variant key KM+H1 is formed as the Exclusive-OR product of master key KM and hash vector H1 and variant key KM+H2 is formed as the Exclusive-OR product of master key KM and control vector H2. In the event that the length of KM differs from the length of H1 and H2, H1 and H2 can be Exclusive-ORed with a portion of KM only. Those skilled in the art will recognize that a combining operation other than the Exclusive-OR operation can be performed instead of the Exclusive-OR operation, without departing from the spirit of the invention. When KM is a DEA master key of 128 bits, then the Exclusive-OR operation calculates the Exclusive-OR product of two 128-bit values, which is the straightforward way in which this operation works. At step 504, the key record supplied as input at 500 is encrypted with variant key KM+H1 to produce the encrypted key record value eKM+H1(key record). Again, those skilled in the art will recognize that many different modes of encryption can be used here, since the goal is to protect the secrecy of the key record but not necessarily to pursue one single strategy for providing an encryption capability. For example, if the variant key KM+H1 is a 64-bit DEA key, then the key record can be encrypted using the Cipher Block Chaining (CBC) mode of encryption. If the variant key KM+H1 is a 128-bit DEA key, then key record can be encrypted using a variation on the CBC mode of encryption. In that case, key record is first encrypted with CBC mode using the leftmost 64 bits of KM+H1, the result is next decrypted with CBC mode using the rightmost 64-bits of KM+H1, and finally that result is encrypted with CBC mode using the leftmost 64-bits of KM+H1. An initialization vector (IV) of zero is used throughout the encryption and decryption operations. In each case, inverse decryption operations are employed in the key record decrypt algorithm, discussed below. Those skilled in the art will recognize that encryption methods other than those illustrated here can be used without departing from the spirit of the invention. At step 505, a hash value HASH2 is calculated on key record using hash algorithm ha2. Hash algorithm ha2 may be different from hash algorithm ha1 or it may be the same. For example, hash algorithm ha2 may be the MDC-2 algorithm of Fig. 5 and HASH2 a 128-bit MDC value. The value HASH2 is for practical purposes defined to be the key authenticator record (KAR). However, the KAR may contain additional data besides HASH2. At step 506, KAR is encrypted with variant key KM+H2 to produce the encrypted KAR value eKM+H2(KAR). Again, those skilled in the art will recognize that many different modes of encryption can be used here, since the goal is to protect the integrity of the KAR by making it infeasible for an adversary to substitute an alternate value of KAR of his or her choosing. Since an adversary has no ability to exercise the encryption function using KM+H2, it is not possible to substitute an encrypted KAR value that will authenticate an encrypted key record, except by mere chance. For example, if the variant key KM+H2 is a 64-bit DEA key, then KAR can be encrypted using the Cipher Block Chaining (CBC) mode of encryption. If the variant key KM+H2 is a 128-bit DEA key, then KAR can be encrypted using a variation on the CBC mode of encryption as described above for the encryption of the key record.

In each case, inverse decryption operations are employed in the key record decrypt algorithm, discussed below. Those skilled in the art will recognize that encryption methods other than those illustrated here can be used without departing from the spirit of the invention. At step 507, the calculated values (a) eKM+H1-(key record) and (b) eKM + H2(KAR) are returned as outputs.

Fig. 17 is block diagram illustration of a first embodiment of the key record decrypt algorithm 13. The first embodiment of the invention covers the case where the cryptographic system implements both a symmetric key algorithm and an asymmetric key algorithm, and where the master key used to encrypt the key records in the key tokens stored outside the cryptographic facility is a symmetric key KM. The key record encrypt algorithm 12 of Fig. 16 and the key record decrypt algorithm 13 of Fig. 17 are inverse algorithms, i.e., key records encrypted with key record encrypt algorithm 12 of Fig. 16 are decrypted with key record decrypt algorithm 13 of Fig. 17. Referring now to Fig. 17, the inputs (a) control vector, (b) eKM+H1(key record), and (c) eKM+H2(KAR) are read at step 510. Control vector is key-related data, or data related to the key stored in key record. Control vector is the same control vector stored in the key token, as described in Fig. 11. eKM+H1(key record) and eKM+H2(KAR) are values produced by the key record encrypt algorithm 12 of Fig. 16. At step 511, a hash value HASH1 is calculated on the control vector using hash algorithm ha1 using the same method as described in step 501 of Fig. 16. At step 512, hash vectors H1 and H2 are calculated from HASH1 using the same method as described in step 502 of Fig. 16. At step 513, variant keys KM+H1 and KM+H2 are calculated from master key KM and hash vectors H1 and H2 using the same method as described in step 503 of Fig. 16. At step 514, the encrypted key record, eKM+H1(key record), supplied as input at 510 is decrypted with variant key KM+H1 to produce the clear value of key record. The method of decryption at step 514 of Fig. 17 is just the inverse operation of encryption at step 504 of Fig. 16. At step 515, a hash value HASH2 is calculated on key record using hash algorithm ha2. Step 515 of Fig. 17 is the same as step 505 of Fig. 16. At step 516, the encrypted KAR, eKM+H2(KAR), supplied as input at 510, is decrypted with variant key KM+H2 to produce the clear value of KAR. The method of decryption at step 516 of Fig. 17 is just the inverse operation of encryption at step 506 of Fig. 16. At step 517, the generated KAR is compared for equality with the decrypted KAR. If equal, then a return code is set equal to "success". If unequal, then a return code is set equal to "failure" and key record is set equal to null (i.e., the recovered key record is erased). At step 518, the values of (a) return code and (b) key record are returned as outputs. If the key record authenticates properly, it is returned as an output at step 518. Otherwise a null value is returned. Those skilled in the art will recognize that there are other ways in which the output values can be returned or not returned. The intent here is for key record decrypt algorithm 13 to return the recovered key record when it authenticates properly and to not return it when it does not authenticate properly. The return code could be omitted from the design, if desired, provided that a protocol is adopted wherein the key record has a special reserved value, say zero, to indicate a failure condition (a nonzero value indicates success).

Fig. 18 is block diagram illustration of a second embodiment of the key record encrypt algorithm 12. The second embodiment of the invention covers the case where the cryptographic system implements an commutative asymmetric key algorithm, and where the master key is an asymmetric key pair (PU0,PR0). Master public key PU0 is used to encrypt key records and to verify digital signatures. Master private key PR0 is used to decrypt key records and to generate digital signatures. Referring now to Fig. 18, the inputs (a) key record and (b) control vector are read at step 520. Key record is the key record to be encrypted and control vector is key-related data, or data related to the key stored in key record. Control vector is the same control vector stored in the key token, as described in Fig. 11. At step 521, the key record supplied as input at 520 is encrypted with public master key PU0 to produce the encrypted key record value ePU0(key record). Since the length of key record may be greater than the block size (or modulus size) of the asymmetric key algorithm, an encryption means must be employed to handle "long" key records. One approach is to use a means similar to Cipher Block Chaining (CBC) mode, as defined for the DEA. In this case, key record is divided into blocks whose length is such that each block can be encrypted with the asymmetric key algorithm. After each step of encryption, the so-produced ciphertext block is Exclusive-ORed with the next block of input plaintext in key record. Those skilled in the art will appreciate that there are many ways in which the encryption with PU0 can be performed and that these various alternate means do not depart from the spirit of the invention. At step 522 control vector and key record are concatenated to form an intermediate value called HA-IN. At step 523, a hash value HASH2 is calculated on HA-IN using hash algorithm ha2. For example, hash algorithm ha2 may be the MDC-2 algorithm of Fig. 5 and HASH2 a 128-bit MDC value. The value HASH2 is for practical purposes defined to be the key authenticator record (KAR). However, the KAR may contain additional data besides HASH2. At step 524, KAR is decrypted with private master key PR0 to produce dPR0(KAR). In public key cryptography, the ciphertext dPR0(KAR) is called a digital signature. In this case, dPR0(KAR) is a digital signature on HA-IN (the concatenation of control vector and key record). At step 525, the calculated values (a) ePU0(key record) and (b) dPR0(KAR) are returned as outputs.

Fig. 19 is block diagram illustration of a second embodiment of the key record decrypt algorithm 13. The second embodiment of the invention covers the case where the cryptographic system implements a commutative asymmetric key algorithm, and where the master key is an asymmetric key pair (PU0,PR0). Master public key PU0 is used to encrypt key records and to verify digital signatures. Master private key PR0 is used to decrypt key records and to generate digital signatures. The key record encrypt algorithm 12 of Fig. 18 and the key record decrypt algorithm 13 of Fig. 19 are inverse algorithms, i.e., key records encrypted with key record encrypt algorithm 12 of Fig. 18 are decrypted with key record decrypt algorithm 13 of Fig. 19. Referring now to Fig. 19, the inputs (a) control vector, (b) ePU0(key record), and (c) dPR0-(KAR) are read at step 530. Control vector is key-related data, or data related to the key stored in key record. Control vector is the same control vector stored in the key token, as described in Fig. 11. ePU0(key record) and dPR0(KAR) are values produced by the key record encrypt algorithm 12 of Fig. 18. At step 531, the encrypted key record, ePU0(key record), supplied as input at 530 is decrypted with private master key PR0 to produce a clear key record. The step of decryption is just the inverse operation of encryption performed at step 521 of Fig. 18. At step 532, control vector supplied as input at 530 and key record recovered at 531 are concatenated to form an intermediate value called HA-IN. Step 532 is just the same as step 522 in Fig. 18. At step 533, a hash value HASH2 is calculated on HA-IN using hash algorithm ha2. The value HASH2 is for practical purposes defined to be the key authenticator record (KAR). However, the KAR may contain additional data besides HASH2. Step 533 is just the same as step 523 in Fig. 18. At step 534, the decrypted KAR, dPR0(KAR), is encrypted with public master key PU0 to produce a clear value of KAR (called the recovered KAR). Note that this is the step that requires the asymmetric key algorithm be commutative. At step 535, the generated KAR is compared for equality with the recovered KAR. If equal, then a return code is set equal to "success". If unequal, then a return code is set equal to "failure" and key record is set equal to null (i.e., the recovered key record is erased). At step 536, the values of (a) return code and (b) key record are returned as outputs. If the key record authenticates properly, it is returned as an output at step 536. Otherwise a null value is returned. Those skilled in the art will recognize that there are other ways in which the output values can be returned or not returned. The intent here is for key record decrypt algorithm 13 to return the recovered key record when it authenticates properly and to not return it when it does not authenticate properly. The return code could be omitted from the design, if desired, provided that a protocol is adopted wherein the key record has a special reserved value, say zero, to indicate a failure condition (a nonzero value indicating success).

Those skilled in the art will recognize that step 521 in Fig. 18 could make use of a decryption operation using the public master key PU0 and step 531 of Fig. 19 could likewise make use of an encryption operation using the private master key PR0. In like manner, step 524 in Fig. 18 could make use of an encrypt operation using private master key PR0 and step 534 of Fig. 19 could make use of a decrypt operation using public master key PU0, as long as both the public key PU0 and private key PR0 remain secret. In fact, the choice of encrypt or decrypt at step 521 of Fig. 18 is independent of the choice of encrypt or decrypt at step 524 of Fig. 18, so that alternate embodiments of the invention can make use of these alternate schemes of encryption versus decryption or decryption versus encryption. And those skilled in the art will recognize that these alternate embodiments do not depart from the spirit of the invention.

Those skilled in the art will also recognize that the key record encrypt algorithm 12 of Fig. 18 and the key record decrypt algorithm 13 of Fig. 19 could make use of a symmetric master key KM instead of a public and private master key (PU0,PR0). In that case, all operations performed with PU0 and PR0 are instead performed with KM. In an alternate approach, variant keys KM1 and KM2 (not equal to KM1) can be used as the master key. In this case, KM1 is used in place of PU0 and KM2 is used in place of PR0. This provides a form of cryptographic separation between the encryption and authentication components of the design. Thus, encryption of the key record is performed with KM1 and encryption of the KAR is performed with KM2. Those skilled in the art will appreciate that these alternate embodiments do not depart from the spirit of the invention.

Fig. 20 shows a functional block diagram of the cryptographic facility 30, for recovering a plurality of public and/or private keys from a key token for use in a plurality of public key algorithms, in response to a plurality of diverse cryptographic service requests. In particular, Fig. 20 depicts how two private keys, PR1 and PR2 can be recovered from a key token accessed from the cryptographic key data set CKDS 32 for use in two different public key algorithms, to fulfill two different cryptographic service requests. The first request R1 is to import the encrypted DEA key ePU1(key1), which was encrypted under a first public key PU1, and decrypt the key under the corresponding private key PR1 to obtain key1, using a first public key algorithm A1. The second request R2 is to generate a digital signature from Data2 under a second private key PR2, using a second public key algorithm A2.

The key token is input from the CKDS on line 50 to the key token register 700, with the header portion in the component register 704 and the concatenated control vector CV, encrypted key record eKM+H1-(parse.Ctl.PR1.PR2) and encrypted key authentication record eKM+H2(KAR) in the component register 702. The header in register 704 defines the beginning and ending of the control vector, the encrypted key record and the encrypted key authentication record in register 702. The header register 704 output is connected to a control input of the multiplexor 706, which separates the control vector for output over line 17 to the control vector register 708, which separates the encrypted key record for output to the encrypted key record register 710 and which separates the encrypted key authentication record for output to the encrypted key authentication register 712.

The control vector checker 714 receives the control vector CV from the register 708. If the Import DEA Key request R1 is the cryptographic service request which has been made, then the control vector checker receives R1 and performs the checking operations on CV to ensure that the key record contains a key which is permitted to be applied to this use. If CV satisfies the control vector checker 714, then an enabling signal "ok" is sent to the gate 716, whose data input is connected to the output of the control vector register 708, passing CV to the control vector input of the key record decrypt algorithm 718 and 720. If CV fails to pass the checks by the control vector checker 714, then the process is aborted.

Alternately, if the Generate Digital Signature request R2 is the cryptographic service request which has been made, then the control vector checker receives R2 and performs the checking operations on CV to ensure that the key record contains a key which is permitted to be applied to this use. If CV satisfies the control vector checker 714, then an enabling signal "ok" is sent to the gate 716, whose data input is connected to the output of the control vector register 708, passing CV to the control vector input of the key record decrypt algorithm 718 and 720. If CV fails to pass these checks by the control vector checker 714, then the process is aborted.

The key record decrypt algorithm 13 in the flow diagram of Fig. 17 is executed by the functional blocks 718, 720, 722, 724, and 726 of Fig. 20. Two functional blocks, 718 and 720, are arranged in parallel and are labeled "Key Record Decrypt Algorithm", in Fig. 20, to provide a clear description of the decryption operations on the encrypted key record and on the encrypted key authentication record. However, in the preferred embodiment of the invention, the two functional blocks 718 and 720 would be combined into a single Key Record Decrypt Algorithm which would operate sequentially on the encrypted key record and on the encrypted key authentication record. Doing so enables second hash vector H2 to be produced from first hash vector H1 by changing only a single bit in H1. The key record decrypt algorithm 718 receives CV and performs the hashing operation described in steps 511 and 512 of Fig. 17, producing the hash vector H1. The master key KM is input from register 15 and the exclusive OR product with H1 is formed, yielding the variant key KM+H1, as described in step 513 of Fig. 17. The second key record decrypt algorithm 720 receives CV and performs the hashing operation described in steps 511 and 512 of Fig. 17, producing the second hash vector H2. The master key KM is input from register 15 and the exclusive OR product with H2 is formed, yielding the second variant key KM+H2, as described in step 513 of Fig. 17. The first key record decrypt algorithm 718 then uses the variant key KM+H1 to decrypt the encrypted key record, as described in step 514 of Fig. 17, yielding the key record (parse,Ctl,PR1,PR2). The key record from key record decrypt algorithm 718 is input to the hash algorithm 724, to produce the computed key authentication record (KAR), as described in step 515 of Fig. 17. Then the computed key authentication record (KAR) is input to a first side of the comparator 726. The second key record decrypt algorithm 720 uses the variant key KM+H2 to decrypt the encrypted key authentication record, as described in step 516 of Fig. 17, yielding the key authentication record KAR. Then the key authentication record KAR is input to a second side of the comparator 726. If the comparator 726 determines that the computed (KAR) is equal to the decrypted KAR, then an enabling signal "yes" is output to a control input of the gate 722, to pass the key record (parse.Ctl.PR1.PR2) from the first key record decrypt algorithm 718 to the key record register 728.

The key record is input to the key record register 728 over line 18, with the parse data in a first component register 730 and the concatenated control information Ctl, first private key PR1 and second private key PR2 in a second component register 732. The parse data in register 730 defines the beginning and ending of the control information Ctl, the first private key PR1 and the second private key PR2 in register 732. The parse data register 730 output is connected to a control input of the multiplexor 734, which separates the control information Ctl for output through register 736 to the public key algorithms 10 and 10', which separates the first private key PR1 for output through register 738 to the gate 742 and which separates the second private key PR2 for output through register 740 to the gate 744.

Gate 742 has a control input connected to receive the Import DEA Key request signal R1, which enables the passing of the first private key PR1 to the first public key algorithm A1 at 10. The encrypted

DEA key ePU1(key1) which was encrypted under a first public key PU1. is input to the operand input of the first public key algorithm A1. The control information Ctl input to the first public key algorithm A1 describes the key type for the first private key PR1 (i.e., specifies PR1 is a decryption key). Using the first private key PR1. the public key algorithm A1 at 10 decrypts the encrypted DEA key ePU1(key1), which was encrypted under a first public key PU1. to obtain the clear text key1.

Gate 744 has a control input connected to receive the Generate Digital Signature request signal R2. which enables the passing of the second private key PR2 to the second public key algorithm A2 at 10'. The clear text Data2 expression is input to the operand input of the second public key algorithm A2. The control information Ctl input to the second public key algorithm A2 describes the key type for the second private key PR2 (i.e.. specifies PR2 is a decryption key). Using the second private key PR2, the public key algorithm A2 at 10' "decrypts" the clear text Data2 expression to obtain the requested digital signature.

Alternate embodiments of the functional block diagram of Fig. 20 can include providing a single key record decrypt algorithm which sequentially performs the functions of algorithms 718 and 720. Another alternate embodiment can include providing a single public key algorithm which sequentially performs the functions of algorithms 10 and 10'. Another alternate embodiment can include storing Key1 in a key block and receiving and processing the key in the encrypted form ePU1(key block). In that case, the output from public key algorithm A1 is a key block containing Key1. Another alternate embodiment eliminates the control information in the key record specifying that the key is a private key or a public key. Instead, the public key algorithms A1 and A2 include a control line indicating encryption or decryption, which is set by cryptographic facility 30 on the basis of the type of cryptographic operation requested. For example, for requests R1 and R2, cryptographic facility 30 will know that the key record contains a private key and that decryption with the private key is required. Thus, a decryption signal can be sent on the control line to the public key algorithms, A1 and A2.

Key Tokens and Key Units: Thus far the described invention has taught that a key token is produced within the cryptographic facility (CF) 30 from a control vector and a key record, as shown in Fig. 21, and the so-produced key tokens are stored outside CF 30 in a cryptographic key data set 32. Referring to Fig. 21, a key record 401 and associated control vector 400 are stored either in an internal key token 403 or an external key token 404. That is, a key token is either an internal key token (also referred to as a key token, i.e., without the modifier 'internal') or an external key token. An Internal Key Token 403 consists of a header 405, a control vector 406. and encrypted key record 407, and an encrypted authenticator 408. The encrypted key record 407 and encrypted authenticator record 408 are produced via key record encrypt algorithm 402, using as inputs control vector 400 and key record 401. Control vector 406 in internal key token 403 is just a copy of control vector 400, which is the control vector associated with key record 401. Key record encrypt algorithm 402 is the same key record encrypt algorithm 12 of Fig. 15. An External Key Token 404 consists of a header 409, a control vector 410, and a key record 411 (i.e., a clear key record). Control vector 410 in external key token 404 is just a copy of control vector 400, which is the control vector associated with key record 401. A key record is either a public key record (i.e., PU key record) or a private key record (i.e., PR key record). Likewise, an internal key token is either a internal PU key token or a internal PR key token, depending on whether the key token contains a PU key record or a PR key record, respectively, and an external key token is either an external PU key token or an external PR key token, depending on whether the key token contains a PU key record or a PR key record, respectively.

However, it may be advantageous to permit the cryptographic facility access program (CFAP) 34 to store key-related information in the key token, not directly available to the CF 30 and therefore not convenient or possible for the CF 30 to store in the key token. Thus, it may be more practical for the CFAP 34 to add certain information fields to the key token once the key token is returned to the CFAP 34 as an instruction output. In such situations where the CFAP is permitted to add information to the key token, a new set of terminology is introduced, as illustrated in Fig. 22. Thus, the internal key token 403 in Fig. 21 becomes internal key unit 423 in Fig. 22. and external key token 404 in Fig. 21 becomes external key unit 435 in Fig. 22. Likewise, control vector 400, key record 401, and key record encrypt algorithm of Fig. 21 are just control vector 420, key record 421, and key record encrypt algorithm 422 of Fig. 22. Likewise, header 405, control 406, encrypted key record 407 and encrypted authenticator record 408 of Fig. 21 are just header 425, control vector 426, encrypted key record 427, and encrypted authenticator record 423 of Fig. 22. Likewise header 409, control vector 410 and key record 411 of Fig. 21 are just header 429, control vector 430 and key record 431 of Fig. 22. Referring again to Fig. 22, internal key token 434 contains IKU 423 as well as other data 432 supplied by CFAP 34. Likewise, external key token 435 contains EKU 424 as well as other data 433 supplied by CFAP 34. Where convenient, the terminology IKU (i.e., internal key unit) and EKU (i.e., external key unit) will be used in lieu of internal key token and external key token when it is necessary to refer to quantities produced by CF 30.

Public Key Cryptographic Design: Full features and apparatus of the invention, which is referred to herein as the Public Key Cryptographic Design (PKCD), are now described. The reader will appreciate that the methods used for key record encryption and decryption described earlier are essential for coupling the usage control to a key in a public key cryptosystem. The reader will also notice that although alternate embodiments have been discussed earlier for key record encryption and decryption, only the first embodiment of Fig. 16 and Fig. 17 is incorporated in the PKCD.

COMPONENTS OF THE CRYPTOGRAPHIC FACILITY

The Cryptographic Facility contains three major components:

o Instruction Processor
o Cryptographic Algorithms
o CF Environment

INSTRUCTION PROCESSOR

Fig. 23 is a block diagram illustration of the components of the Instruction Processor. They are:

o INSTRUCTIONS: The CF instructions are invoked at the CFAP-To-CF interface. They provide the following cryptographic services to the CFAP:
System Digital Signatures
Application Digital Signatures
Key Management
CKDS Update
CF Backup
CF Audit
CF Initialization
CF Configuration
CF Control
Utility

o INTERNAL ROUTINES: The internal routines are invoked only from within the CF. Collectively they represent a set of algorithms and processing functions that are common to many CF instructions. The internal routines have been specified to simplify the architectural description and definition, and to make each instruction's functional specification precise and less apt to contain errors and ambiguities. Although the internal routines are an integral part of the instruction functional specifications, an implementer may elect to implement the instructions and internal routines in a way that best suits or optimizes the particular implementation.

o CONFIGURATION TABLE: The Configuration Table is a collection of constants that may vary in value from one implementation to another. The Configuration Table permits the Instructions and Internal Routines to be defined in a more general and open-ended way. Unlike the CF Environment, the Configuration Table is an integral part of the CF (e.g., hardware or ROS microcode).

Fig. 24 is a block diagram illustration of the elements in the Configuration Table.

CRYPTOGRAPHIC ALGORITHMS

Fig. 25 is a block diagram illustration of the main components of Cryptographic Algorithms of the CF. The Cryptographic Algorithms components are these:

o DATA ENCRYPTION ALGORITHM (DEA): The DEA is described in the American National Standards Institute (ANSI) Data Encryption Algorithm (DEA) X3.92-1981. The DEA is a symmetric algorithm which encrypts or decrypts a 64 bit input with a 64 bit key to produce a 64 bit output. The 64 bit key specified to the algorithm consists of 56 key bits used by the algorithm and 8 non-key bits, which optionally may be used for error detection. According to ANSI X3.92-1981, the 8 non-key bits MAY be used for error detection.. On the other hand, according to FIPS PUB 46, the 8 non-key bits SHALL be used for error detection and more specifically the error detection is based on byte-by-byte odd parity. Although the Symmetric Key Cryptographic Algorithm can be an optional component of the Cryptographic algorithms 144 shown in Fig. 15, the DEA is a required component in the PKCD, as it is needed for key record encryption and decryption.

o PUBLIC KEY ALGORITHM (PKA): PKA is a generic term referring to one of several possible public key algorithms. The PKCD does not specify the use of a particular PKA. However, the PKA must permit key distribution to be based on a key server concept wherein a DEA key, randomly generated and encrypted with a public key of a receiving device, is served to a receiving device where it is decrypted with the private key of the receiving device and reencrypted under the master key. The PKA must also permit generation and verification of digital signatures. A digital signature is produced by decrypting a signature record, containing a hash value, with a private key. A digital signature is verified by encrypting the signature with a public key and comparing hash values. The PKCD also permits key distribution with a first PKA and digital signatures to be implemented with a second PKA.

o PUBLIC KEY ALGORITHM KEY GENERATOR (PKAKG): PKAKG is a separate algorithm for the generation of keys used by the PKA.

Besides the main components, there are lower level algorithms, such as Key Record Encryption and Key Record Decryption algorithms needed for frequent encryption and decryption of public and private keys, as discussed earlier.

CF ENVIRONMENT

Fig. 26 is a block diagram illustration of the components of the CF Environment.

The CF Environment components are these:

o CONFIGURATION VECTOR: The configuration vector is a collection of encoded fields that limit or restrict the operation of the cryptographic facility. The configuration vector is set to a default value via execution of the Enter Initialization State (EIS) instruction, or it may be set to an installation-specified value via execution of the Load Configuration Vector (LCV) instruction.

o STATE VECTOR: The state vector is a collection of flags and state variables that define the current state of the cryptographic facility. The state vector is used by the instruction processor to control the order in which PKCD instructions are executed.

o REGISTERS: The registers contain space for the storage PKCD cryptovariables, including keys, MDC values, internal counters, identifiers, and control vectors.

o MDC TABLE: The MDC table contains space for the storage of Modification Detection Codes (MDCs) used by the Import Public Key (IPUK) instruction to import External Key Units. Each table entry is an MDC calculated on an External Key Unit using a hash algorithm.

o COUNTER TABLE: The Counter table contains space for the storage of counters, where each counter is associated with a particular PKCD instruction. Counter(i) contains a value "n" from 1 to 255, set by the SEF instruction, which represents the number of times instruction "i" is permitted to be executed.

o CFPKR1-LENGTH: The length of cfpkr1 in 8-byte blocks. cfpkr1 is stored in the PUA Buffer and contains the Public Device Authentication Key (PUA).

o PUA BUFFER: The PUA buffer contains space for the storage of cfpkr1, which contains PUA. The PUA buffer is used only by the PKCD instructions.

o CFPKR2-LENGTH: The length of cfpkr2 in 8-byte blocks. cfpkr2 is stored in the PRA Buffer and contains the Private Device Authentication key (PRA).

o PRA BUFFER: The PRA buffer contains space for the storage of cfpkr2, which contains PRA. The PRA buffer is used only by the PKCD instructions.

o SECRET PRODUCT ENVIRONMENT LENGTH: The length of the secret product environment in bytes.

o SECRET PRODUCT ENVIRONMENT: The secret product environment consists of a set of the secret cryptographic variables unique to a product or implementation. That is, secret cryptographic variables not specified by PKCD but needed by a product.

o NONSECRET PRODUCT ENVIRONMENT LENGTH: The length of the nonsecret product environment in bytes.

o NONSECRET PRODUCT ENVIRONMENT: The nonsecret product environment consists of a set of the nonsecret cryptographic variables unique to a product or implementation. That is, nonsecret cryptographic variables not specified by PKCD but needed by a product.

o EKU LENGTH: The length in bytes of the EKU in the EKU buffer.

o EKU BUFFER: A buffer for the temporary storage of an External Key Unit (EKU) (e.g., an EKU loaded into the CF via an interface other than the CFAP-to-CF interface).

o GKSP SAVE: A field used by process-mode = 1 of the Generate Key Set Pair (GKSP) instruction to save information needed by process-mode = 2 of the GKSP instruction.

o GKSP BUFFER LENGTH: The length of GKSP Buffer in bytes.

o GKSP RECORD LENGTH: The length of record or block in GKSP Buffer in bits.

o GKSP BUFFER FLAG: A flag indicating the status of the record or block in GKSP Buffer, as follows:

- 4-255 : reserved
- 3 : GKSP Buffer contains a record of unspecified format that must be processed to produce a keyblk which is then encrypted.
- 2 : GKSP Buffer contains a keyblk of unspecified format that needs only to be encrypted.
- 1 : GKSP Buffer contains a CF DEA Key Record.
- 0 : GKSP Buffer is empty

o GKSP TICKET: An 8-byte pseudorandom value generated via execution of process-mode = 1 of the GKSP instruction.

o GKSP BUFFER: A buffer for the storage of a key record or key block.

o IDK SAVE: A field used by process-mode = 1 of the Import DEA Key (IDK) instruction to save information needed by process-mode = 2 of the IDK instruction.

o IDK BUFFER LENGTH: The length of IDK Buffer in bytes.

o IDK RECORD LENGTH: The length of record or block in IDK Buffer in bits.

o IDK BUFFER FLAG: A flag indicating the status of the record or block in IDK Buffer, as follows:

- 4-255 : reserved
- 3 : IDK Buffer contains a record of unspecified format recovered from a keyblk of specified format recovered by process-mode = 1 of the IDK instruction by decrypting ePUM(keyblk).
- 2 : IDK Buffer contains a keyblk of unspecified format recovered by process-mode = 1 of the IDK instruction by decrypting ePUM(keyblk).
- 1 : IDK Buffer contains a CF DEA Key Record.
- 0 : IDK Buffer is empty

o IDK TICKET: An 8-byte pseudorandom value generated via execution of process-mode = 1 of the IDK instruction.

o IDK BUFFER: A buffer for the storage of a key record or key block.

CONFIGURATION VECTOR

The configuration vector has the following specification:

CONFIGURATION VECTOR:

bits

00.. 07 Version Number

X'00' : reserved
X'01' : PKCD
X'10 - X'FF' : reserved

08..151 DEFINE field

The DEFINE field is a vector indexed as DEFINE(i) for i = 0,1,...,143.
For i = 0,109 DEFINE(i) is reserved.
For i = 110,111,...,143 DEFINE(i) pertains to the instructions of the PKCD.
DEFINE(i) for i = 110,...,143 has the following meaning:
B'1' : instruction is defined to the CF in the "run" state
B'0' : instruction is not defined to CF in the "run" state
Note: DEFINE(i) for i = 110,...,143 pertains only to execution of instructions in the "run" state.
A list of the instructions and their corresponding indices are provided in Fig. 27.

152..295 AUTH CONTROL field

The AUTH CONTROL field is a vector indexed as AUTH CONTROL(i) for i = 0,1,...,143.
For i = 0,1, ...,109 AUTH CONTROL(i) is reserved.
For i = 110, ...,143 AUTH CONTROL(i) pertains to the instructions of PKCD.
AUTH CONTROL(i) has the following meaning:
B'1' : the LCV instruction sets AUTH(i) = B'1' and ENABLE(i) = B'11' (i.e., "authorization required" & "disabled").

B'0' : the LCV instruction sets AUTH(i) = B'0' and ENABLE(i) = B'00' (i.e., "authorization not required" & "enabled").

A list of the instructions and their corresponding indices are provided in Fig. 27.

296 CERTIFICATION

B'1' : certification center (the device can act as a certification center)

B'0' : not a certification center (the device cannot act as a certification center). This means the following: Generate Public and Private Key Pair (GPUPR) cannot generate a certification key pair; a PRC key cannot be used with the Generate Digital Signature (GDS), Generate Application Digital Signature (GADS), and/or Export Public Key (EPUK) instructions to generate a digital signature.

297 KMP RELOAD

B'1' : if CKMP HISTORY in state vector = 0, then KMP-mode = 1 must be specified in the ECFER instruction (i.e., the PKA Key Encrypting Master Key (KMP) must be reloaded at the receiving device).

B'0' : no restrictions Note that this field pertains only to the ECFER instruction.

298 KM RELOAD

B'1' : (reserved for future use) if CKM HISTORY in state vector = 0, then KM-mode = 1 must be specified in the ECFER instruction (i.e., the DEA key encrypting master key KM must be reloaded at the receiving device).

B'0' : no restrictions Note: this field pertains only to the ECFER instruction. Note: the LCV instruction sets this bit = B'0', which guarantees that present systems shall be compatible with future releases implementing the KM RELOAD bit.

299..300 FLOOR-MDC field

The FLOOR-MDC field specifies the following:

a. The minimum THRES-MDC value that may be specified in the PRM control vector in the GPUPR instruction.

b. The minimum HIST-MDC value in the PUA control vector that can be processed by the ECFER and ICFER instructions.

The FLOOR-MDC field has the following meaning:

B'11' : The referenced THRES-MDC or HIST-MDC must have a value = B'11.

B'10' : The referenced THRES-MDC or HIST-MDC must have a value ≥ B'10.

B'01' : The referenced THRES-MDC or HIST-MDC must have a value ≥ B'01.

B'00' : reserved

Note that the FLOOR-MDC field controls the processing of PU keys in the GPUPR, ECFER, and ICFER instructions.

301..302 KMGT PROTOCOL (i.e., key management protocol via the GKSP and IDK instructions).

B'11' : CKMGT & PKMGT (i.e., certification center and private key management protocols are enabled).

B'10' : CKMGT (i.e., certification center key management protocol is enabled)

B'01' : PKMGT (i.e., private key management protocol is enabled)

B'00' : none

Notes:

a) KMGT PROTOCOL = B'11' means that the key-management-protocol parameter in the GKSP and IDK instructions may be 0 or 1.

b) KMGT PROTOCOL = B'10' means that the key-management-protocol parameter in the GKSP and IDK instructions may only be 1.

c) KMGT PROTOCOL = B'01' means that the key-management-protocol parameter in the GKSP and IDK instructions may only be 0.

d) KMGT PROTOCOL = B'00' means that key management via the GKSP and IDK instructions is not permitted.

303..304 BKUP PROTOCOL (i.e., protocol for CF-environment backup via the ECFER and ICFER instructions).

B'11' : PBKUP (private protocol, i.e., no restriction on how PUA is imported)
B'10' : CBKUP2 (certification center protocol where the PUA control vector has HIST-CHAIN = 3)
B'01' : CBKUP1 (certification center protocol where the PUA control vector has HIST-CHAIN = 2)
B'00' : no backup permitted

Note that the specification matches that of the protocol-mode parameter in the ECFER and ICFER instructions.
The BKUP PROTOCOL field is valid only when DEFINE(ECFER) = B'1' or DEFINE(ICFER) = B'1'.

305 KREG field

The KREG field defines the key registration mode or modes permitted for certification center key management and for certification center backup, as follows:
1 : restricted mode
0 : unrestricted mode

Note: the meanings attached to restricted mode and unrestricted mode are specified by the network, i.e., set forth according to network security policy.
For example, the certification center could define restricted mode such that the conditions in (a) or (b) must be satisfied, as follows:

(a) PUM key registration is performed in a physically secure environment; KMP is loaded into the device by trusted personnel using the LFPMKP and CPMKP instructions or KMP is internally generated using the GNPMK instruction.

(b) PUM key registration is not performed in a physically secure environment, but the External Key Unit containing PUM (sent to the certification center for registration) is signed with a PRA key which has been independently validated by the certification center as originating from within the said device. KMP is internally generated using the GNPMK instruction.

Both (a) and (b) represent very 'high security' modes.

306 INTERCHANGE

B'1' : interchange (the device can act as an interchange device) A PRA, PRM, and PRU key can be used with the GDS instruction to generate digital signatures.
B'0' : not interchange (the device cannot act as an interchange device). A PRA, PRM, and PRU key cannot be used with the GDS instruction to generate digital signatures.

307..308 SIG-COMPATIBILITY field

The SIG-COMPATIBILITY field is a vector indexed as COMPATIBILITY(i) for i = 0 and 1.
For i = 0, SIG-COMPATIBILITY(i) pertains to the IPUK instruction.
For i = 1, SIG-COMPATIBILITY(i) pertains to the IDK instruction.
SIG-COMPATIBILITY(i) has the following meaning:
B'1' : the instruction does not require CF authentication of system signatures.
B'0' : the instruction requires CF authentication of system signatures.

309..511 reserved, set := 203 B'0'.

DEFAULT CONFIGURATION VECTOR

The default configuration vector is the value of the configuration vector automatically set via execution of an EIS instruction. The value of the default configuration vector in 8 groups of 16 hexadecimal digits per group is as follows:

1. X'01FFFFFFFFFFFFFF'
2. X'FFFF0000000003FF'

3. X'FFFFFF0000000000'
4. X'0000000000000000'
5. X'00000000000E0000'
6. X'0000000000000000'
7. X'0000000000000000'
8. X'0000000000000000'

The default configuration vector has the following specification:

DEFAULT CONFIGURATION VECTOR:

bits	value	field
00.. 07	(= B'00000001')	Version Number
08..151		DEFINE
08.. 79	(= 72 B'1')	Reserved
80..117	(= 38 B'0')	reserved
118..151	(= 34 B'1')	PKCD instructions (VADS thru VIKU)
152..295	(= 144 B'0')	AUTH CONTROL
296	(= B'0')	CERTIFICATION (not certification center)
297	(= B'0')	KMP RELOAD (no restrictions)
298	(= B'0')	KM RELOAD (no restrictions)
299..300	(= B'01')	FLOOR-MDC field (The referenced THRES-MDC or HIST-MDC must have a value ≥ B'01'.)
301..302	(= B'11')	KMGT PROTOCOL (CKMGT and PKMGT modes)
303..304	(= B'00')	BKUP PROTOCOL (backup not permitted)
305	(= B'0')	KREG field (unrestricted mode)
306	(= B'0')	INTERCHANGE (not an interchange)
307	(= B'0')	SIG-COMPATIBILITY(IPUK) (signature required)
308	(= B'0')	SIG-COMPATIBILITY(IDK) (signature required)
309..511	(= 203 B'0')	reserved

STATE VECTOR

The state vector has the following specification:

STATE VECTOR:

00 KP FLAG (Key Part)

B'1' : the KP register is in the "full" state
B'0' : the KP register is in the "empty" state

01 OKM FLAG (Old DEA key encrypting master key)

B'1' : the OKM register is in the "full" state
B'0' : the OKM register is in the "empty" state

Note: In a subsequent release of PKCD, the existing OKM flag shall be reimplemented within the state vector. For the present, this field is initialized only via the ECFAR instruction (thus making it appear that the OKM flag is implemented within the state vector).

02 CKM FLAG (Current DEA key encrypting master key)

B'1' : the CKM register is in the "full" state
B'0' : the CKM register is in the "empty" state

Note: In a subsequent release of PKCD, the existing CKM flag shall be reimplemented within the state vector. For the present, this field is initialized only via the ECFAR instruction (thus making it appear that the CKM flag is implemented within the state vector).

03.. 04 NKM FLAG (New DEA key encrypting master key)

B'11' : reserved
B'10' : the NKM register is in the "full" state
B'01' : the NKM register is in the "partially full" state
B'00' : the NKM register is in the "empty" state

Note: In a subsequent release of PKCD, the existing NKM flag shall be reimplemented within the state vector. For the present, this field is initialized only via the ECFAR instruction (thus making it appear that the NKM flag is implemented within the state vector).

05 Reserved

06.. 12 RESERVED (= 7B'0')

13 PROG FLAG

B'1' : An executable program has been loaded
B'0' : An executable program has not been loaded

14 PROGMDC0 FLAG (Secure loadable program MDC #0)

B'1' : PROGMDC0 buffer is in the "full" state
B'0' : PROGMDC0 buffer is in the "empty" state

15 PROGMDC1 FLAG (Secure loadable program MDC #1)

B'1' : PROGMDC1 buffer is in the "full" state
B'0' : PROGMDC1 buffer is in the "empty" state

16.. 21 KM HISTORY field

The KM HISTORY field is a vector indexed as KM HISTORY(i) for i = 0,1,2, where i is defined as:
0 : OKM
1 : CKM
2 : NKM

KM HISTORY(i) has the following meaning:

B'11' :	reserved
B'10' :	(reserved) GNDMK (i.e.. the contents of the KM register were produced via execution of the GNDMK instruction).
B'01' :	(reserved) LFMKP/CMKP (i.e.. the contents of the KM register were produced via execution of the LFMKP and CMKP instructions).
B'00' :	indeterminate

Note: The EPS instruction sets this field to B'000000', ensuring the present PKCD will be compatible with future releases of PKCD implementing code points B'01' and B'10' of the KM HISTORY field.

Note: In a subsequent release of PKCD, the SMK instruction shall be modified so that the OKM HISTORY field is updated from the CKM HISTORY field.

Note: In a subsequent release of PKCD, the SMK instruction shall be modified so that the CKM HISTORY field is updated using a method which is similar to that followed by the SPMK instruction in its management of the CKMP HISTORY field. The SMK instruction shall also be modified so that the CKM HISTORY field is reset to the "indeterminate" state whenever the CKM FLAG is reset to the "empty" state, and the NKM HISTORY field is reset to the "indeterminate" state whenever the NKM FLAG is reset to the "empty" state.

Note: In a subsequent release of PKCD. the GNDMK instruction shall be modified to reset the NKM HISTORY field to B'10', and the LFMKP and CMKP instructions shall be modified to reset the NKM HISTORY field to B'01'.

22..165 AUTH field

The AUTH field is a vector indexed as AUTH(i) for i = 0,1,...143.
For i = 0,1, ...,109 AUTH(i) is reserved
For i = 110, ...,143 AUTH(i) pertains to instructions of the PKCD.
AUTH(i) has the following meaning:

B'1' :	the SEF instruction can be used to enable execution of instruction or instruction mode "i" only after supplying appropriate proof of authorization to the CF. The level of authorization is determined by the implementation and can be different for each "i."
B'0' :	no restrictions

AUTH(i) is not defined for the following instructions:
(1) LPID, GDAK, LCV, since they do not execute in the "run" state.
(2) ERS, since the specification is contradictory.
(3) SEF, since this could lead to "lockout."

A list of the instructions and their corresponding indices are provided in Fig. 27.

166..453 ENABLE field

The ENABLE field is a vector indexed as AUTH(i) for i = 0,1,...143.
For i = 0,1, ...,109 ENABLE(i) is reserved
For i = 110, ...,143 ENABLE(i) pertains to instructions of the PKCD.
ENABLE(i) has the following meaning:

B'11' :	instruction or instruction mode execution not enabled.
B'10' :	instruction or instruction mode enabled for n executions, where n (a value from 1 to 255) is a value specified by an instruction input parameter.
B'01' :	instruction or instruction mode enabled for 1 execution.
B'00' :	instruction or instruction mode enabled for any number of executions.

ENABLE(i) = B'00' and ENABLE = B'11' are valid for all but the following instructions:
(1) LPID, GDAK, LCV, since they do not execute in the "run" state.
(2) ERS, since the specification is contradictory.
(3) SEF, since this could lead to "lockout."

ENABLE(i) = B'10' is valid only for the following instructions:
(1) CPMKP input-mode = 0 and CPMKP input-mode = 1
(2) GPUPR mode = 0/2.

ENABLE(i) = B'01' is valid only for the following instructions:
(1) LMDCC
(2) LMDC
(3) LFPMK input-mode = 0 and LFPMK input-mode = 1

(7) GNPMK,
(8) GNDMK.
(9) SPMK,
(10) ECFER

A list of the instructions and their corresponding indices is provided in Fig. 27.

454 CLONE (history bit)

B'1' : CF-environment has been set via the ICFER instruction.
B'0' : original CF-environment

Note: this bit is reset to 0 via execution of an EPS instruction but not by an EIS instruction. This bit is set to B'1' via execution of the ICFER instruction.

455..457 KMP-HISTORY field

The KMP-HISTORY field is a vector indexed as KMP-HISTORY(i) for i = 0,1,2, where i is defined as:

0 : OKMP HISTORY
1 : CKMP HISTORY
2 : NKMP HISTORY

KMP stands for PKA Key encrypting Master key.

KMP-HISTORY(i) has the following meaning:

B'1' : GNPMK (i.e., the contents of the KMP register were produced via execution of the GNPMK instruction).
B'0' : LFPKMP/CPMKP (i.e., the contents of the KMP register were produced via execution of the LFPKMP and CPMKP instructions).

Note: KMP-HISTORY(i) has meaning only when KMP-FLAG(i) is in the "full" state.

458..461 KMP-FLAG field

The KMP-FLAG field is a vector indexed as KMP-FLAG(i) for i = 0,1,2, where i is defined as:

0 : OKMP FLAG
1 : CKMP FLAG
2 : NKMP FLAG

For i = 0 and 1, KMP-FLAG(i) is a 1 bit field with the following meaning:

B'1' : the KMP register is in the "full" state
B'0' : the KMP register is in the "empty" state

For i = 2, KMP-FLAG(i) is a 2 bit field with the following meaning:

B'11' : reserved
B'10' : the KMP register is in the "full" state
B'01' : the KMP register is in the "partially full" state
B'00' : the KMP register is in the "empty" state

462 GDAK FLAG

B'1' : the PUA buffer, PRA buffer, PUACV register, and PRACV register are in the "full" state.
B'0' : the PUA buffer, PRA buffer, PUACV register, and PRACV register are in the "empty" state.

463 LPID FLAG

B'1' : the DID and EID registers are in the "full" state.
B'0' : the DID and EID registers are in the "empty" state.

464 LCV FLAG

B'1' : a configuration vector has been loaded using an LCV instruction.
B'0' : a configuration vector has not been loaded using an LCV instruction.

If CF STATE = "init" or "run", then CONFIG FLAG = B'0' indicates that a default configuration vector has been loaded.

465 EKU FLAG

B'1' : an EKU is stored in the EKU buffer and its length is stored in EKU Length.
B'0' : an EKU is not stored in the EKU buffer and its length is not stored in EKU Length.

466..467 CF STATE

B'11' : reserved
B'10' : the CF is in the "run" state
B'01' : the CF is in the "init" state
B'00' : the CF is in the "pre-init" state
Note: the CF states control instruction execution.

468..501 EKUMDC FLAG field

The EKUMDC FLAG field is a vector indexed as EKUMDC FLAG(i) for i = 0,1,...,16.
For i = 0,1,...,15, EKUMDC FLAG(i) has the following meaning:
B'11' : EKUMDC(i) has been initialized with an MDC loaded via a secure interface (e.g., via a smart card).
B'10' : EKUMDC(i) has been initialized with the LMDCC instruction.
B'01' : EKUMDC(i) has been initialized via the IPUK instruction.
B'00' : EKUMDC(i) is uninitialized.
For i = 16, EKUMDC FLAG(16) has the following meaning:
B'11' : EKUMDC(16) has been initialized with an MDC loaded via a secure interface (e.g., via a smart card).
B'10' : EKUMDC(16) has been initialized via an LMDC instruction.
B'01' : reserved
B'00' : EKUMDC(16) is uninitialized.

502 PR HISTORY

B'1' : one or more PR have not been randomly generated inside the CF.
B'0' : all PR have been randomly generated inside the CF.

503 ECFER Status

B'1' : the ECFER instruction has been executed at least once (i.e., the CF environment of this device has been exported).
B'0' : the ECFER instruction has not been executed.

504 ALARM FLAG

B'1' : Alarm has been activated
B'0' : no Alarm

505..508 HIST-DOMAIN

This field contains a domain identifier (an arbitrary value from B'0000' to B'1111'). The HIST-DOMAIN field in the state vector is set by the ECFER instruction equal to the value of DOMAIN ID in the HIST-DOMAIN field of the PUA control vector contained in IKU1, which is input to the ECFER instruction.
This field is valid only if the CLONE bit in the state vector is equal to B'1' and BKUP PROTOCOL in the configuration vector is equal to B'01' (CBKUP1) or B'10' (CBKUP2).

509..511 reserved, set = 3 B'0'.

REGISTERS

The following registers are defined by PKCD:

```
*----------------------*
| NKMP Register      S |  128b  New PKA Master Key Register
|----------------------|
| CKMP Register      P |  128b  Current PKA Master Key Register
|----------------------|
| OKMP Register      P |  128b  Old PKA Master Key Register
|----------------------|
| DID Register       P |  128b  Device Identifier Register
|----------------------|
| EID Register       P |  128b  Environment Identifier Register
|----------------------|
| PUACV Register     P |  128b  PUA Control Vector
|----------------------|
| PRACV Register     P |  128b  PRA Control Vector
*----------------------*
```

The registers are designated as permanent (encoded with letter "P") or semi-permanent (encoded with letter "S"). The contents of the permanent registers must be preserved for the "life of the system," e.g., via a battery-backed RAM. Values stored in the permanent registers change or are changed according to an installation-determined schedule. The contents of the semipermanent registers must be preserved only until the information they contain has been processed by a CF instruction.

MDC TABLE

The MDC Table is a vector EKUMDC(i), for i = 0,1,...,16, where each EKUMDC(i) contains storage for a 128 bit MDC value.

Fig. 28 illustrates the organization and indexing of the MDC Table.

The MDC Table is used by the IPUK instruction to import public keys, which are presented to the IPUK instruction in the form of an External Key Unit (EKU). For i=0,...,15, the MDC in EKUMDC(i) must be calculated on an EKU containing a public certification key (i.e., a PUC key) and the domain ID field in the control vector of the PUC key must contain the value "i". For i=16, the MDC in EKUMDC(i) must be calculated on a EKU containing a public key management key, a public authentication key, or a public user key (i.e., a PUM, PUA, or PUU key). The domain ID field can contain any value from 0 to 15.

COUNTER TABLE

The Counter Table is a vector COUNTER(i), for i = 0,1,...,143, where each COUNTER(i) contains storage for an 8 bit counter. For i = 113, 114, or 120, COUNTER(i) is defined. For i ≠ 113, 114, or 120, COUNTER(i) is not defined (i.e., this portion of Counter Table is null).

The value of COUNTER(i) denotes the number of times that instruction "i" can be executed before ENABLE(i) is set from B'10' to B'11'. Fig. 27 specifies the relationship between index and instruction name. For example, i=113 denotes input-mode=0 of the CPMKP instruction.

Fig. 29 illustrates the organization and indexing of the Counter Table.

CONTROL VECTOR ENFORCEMENT

Control vector enforcement is a method which ensures that the control vectors processed by each CF instruction are consistent with and in conformance with certain instruction-unique rules and restrictions which limit or define the values that these control vectors may have. Control vector enforcement may be accomplished by, although is not limited to, one of the following methods or combinations thereof:

- o SPECIFY CONTROL VECTOR IN CFAP AND CHECK CONTROL VECTOR BITS IN CF: This method checks bits and fields within the control vector to ensure that they contain permitted values. In certain cases, cross checking of bits and fields among two or more control vectors is necessary to ensure that they contain only permitted combinations of values.
- o SPECIFY CONTROL VECTOR IN CFAP AND SET CONTROL VECTOR BITS IN CF: This method sets bits and fields within the control vector to prescribed values (i.e., by overwriting the bits and fields of the control vectors passed at the instruction interface).
- o GENERATE CONTROL VECTOR IN CF FROM INFORMATION SPECIFIED BY CFAP: This method generates control vectors from parameter information passed at the instruction interface.
- o TABLE LOOKUP OF CONTROL VECTOR IN CF FROM INDEX SPECIFIED BY CFAP: This method uses a table of control vectors stored within the CF. An index value passed at the instruction interface selects the control vector or vectors used by an instruction.

For convenience, control vector enforcement is defined in this teaching using a combination of the first and third methods described above. Some control vectors are specified as instruction parameters and bits and fields in these control vectors are checked by the CF. Other control vectors are generated within the CF, e.g., it is typical for the control vector associated with the rightmost 64 bits of a 128 bit key to be derived from the control vector associated with the leftmost 64 bits of a 128 bit key.

INITIALIZATION REQUIREMENTS

Some CF instructions process cryptovariables stored internally within the CF, which must be loaded or imported into the CF before instruction execution. Several CF instructions have been defined to support the initialization and configuration of the CF. However, PKCD does not define or specify how key parts are loaded into the KP register.

Those CF instructions which process cryptovariables stored in the KP register, which must be loaded via means other than those defined in the PKCD, are listed in the following table.

Instruction	Cryptovariable	CF Storage Location
LFPMKP	key part	KP register
CPMKP	key part	KP register

PKCD do not define how key parts are loaded into the KP register. One possibility is for key parts to be loaded by authorized installation personnel via a protected, controlled interface. The physical interface described earlier could be used for this purpose.

POWER ON SEQUENCE

During each power-on, the CF executes a power-on sequence (POS) routine. The POS routine does the following:

- o Initialize the PRNGKEY1 and PRNGKEY2 registers with random seed keys.
- o If the content of the POS register = X'0123456789ABCDEF0123456789ABCDEF' then continue; else do the following:
  - Perform the EIS instruction to clear the CF environment.
  - Set POS register := X'0123456789ABCDEF0123456789ABCDEF'.

RECORD FORMATS AND DESCRIPTIONS

The following records are defined by PKCD:

Record Name	Length
Crypto Facility PKA Key Record	multiple of 8 bytes
Crypto Facil.Key Authenticator Record	multiple of 8 bytes
Crypto Facility DEA Key Record	64 bytes
Crypto Facility Backup DEA Key Record	64 bytes
Crypto Facil.System Signature Record	64 bytes
Crypto Facility Environment Record	multiple of 8 bytes
Crypto Facility Audit Record	multiple of 8 bytes
Internal Key Unit	multiple of 8 bytes
Clear Key Unit	multiple of 8 bytes
External Key Unit	multiple of 8 bytes
Skeleton Key Unit	multiple of 8 bytes

CRYPTO FACILITY PKA KEY RECORD (CFPKR)

The Crypto Facility PKA Key Record (CFPKR) contains a public or private key used with a public key algorithm. If different public key algorithms are used for key distribution and digital signatures, then the CFPKR contains two public or two private keys--one key for key distribution and the other for digital signatures. The CFPKR is defined to be a multiple of 8 bytes.

A Crypto Facility PKA Key Record has the following form:

Offset	Length	Data
(in bits)		
0	a	Parse
a	b	Key
c	d	RN (where c = a+b)
e	0	End of CFPKR (where e = c+d)

DATA	DESCRIPTION
PARSE	The parse field contains data that permits the CF to parse the key field. The length of the key parse field is not prescribed by the architecture. The key parse field MUST permit the key, or any portion of the key, to be uniquely identified and located in the key field. The parse field must directly or indirectly specify key length, such that an adversary cannot cause the CF to use only a portion of a key as a full key. In addition, the parse field contains at least 8 bytes of random data to act as a confounder to thwart revealing any contents of the encrypted CFPKR by pattern analysis by an adversary.
KEY	The key field contains a PKA key. The key is either a public key or a private key. The key stored in the key field consists of one or more key variables that together constitute or define the key. For example, if the PKA is based on exponentiation modulo a number n, then the key consists of an exponent e and a modulus n. Both e and n are stored in the key field, and the parse field is defined in such a way to permit e and n to be located. Key length and format of the parse and key fields can be different depending on whether the key is a public key or a private key. If a first PKA is used for key distribution and a second PKA is used for digital signatures, then the Key field contains a pair of public or private keys (i.e., a key for each algorithm). The fact that there are two algorithms is made transparent to the CFAP.
RN	The RN field contains a d-bit random number generated within the CF. The value d ranges from 0 to 63 and is chosen so that the length of CFPKB is a multiple of 8 bytes.

Outside the CF, the CFPKR is encrypted under a variant key KMP.C formed as the Exclusive OR product of KMP and control vector C.

CRYPTO FACILITY KEY AUTHENTICATOR RECORD (CFKAR)

The Crypto Facility Key Authenticator Record (CFKAR) contains information functionally related to a single CFPKR. The CFKAR is used to authenticate a CFPKR. The CFKAR is defined to be a multiple of 8 bytes.

A Crypto Facility Key Authenticator Record has the following form:

Offset	Length	Data
(in bits)		
0	a	Key Authenticator
a	b	RN
c	0	End of CFPKR (where c = a+b)

DATA	DESCRIPTION
KEY AUTHENT!CATOR	The key authenticator field contains data functionally related to a CFPKR.
RN	The RN field contains a b-bit random number generated within the CF. The value b ranges from 0 to 63 and is chosen so that the length of CFKAR is a multiple of 8 bytes.

Outside the CF, the CFKAR is encrypted under a variant key KMP.C formed as the Exclusive OR product of KMP and control vector C.

Methods for deriving a key authenticator from a key record has been discussed in Key Record Encrypt Algorithm 12 of Fig. 16.

CRYPTO FACILITY DEA KEY RECORD (CFDKR)

The Crypto Facility DEA Key Record is produced by a GKSP instruction and is processed by an IDK instruction. The CFDKR is a 52 byte record.

A Crypto Facility DEA Key Record has the following form:

Offset	Length	Data
(in bytes)		
0	1	Record ID The most significant bit in a byte is the leftmost bit. 0000 0000 - 'Crypto Facility DEA Key Record'
1	1	Record Code 000 xxxxx - 128b key-encrypting key produced by GKSP and processed by IDK. When bits 0..2 of the above field are B'000', bits 3..7 are defined as follows: Control Vector Format 000 xxx00 - control vector field stores hash of 128 bit C. 000 xxx01 - control vector field stores hash of 64 bit C. 000 xxx1x - reserved KEY-MANAGEMENT-PROTOCOL specified in GKSP (implying IDK must also specify the same) 000 0xxxx - private protocol 000 1xxxx - certification center protocol KEY-MANAGEMENT-MODE specified in GKSP (implying IDK must also specify the same) 000 x0xxx - key registration is performed using mode 0 000 x1xxx - key registration is performed using mode 1 001 xxxxx - reserved 01x xxxxx - reserved 1xx xxxxx - reserved
2	2	Reserved (=X'0000')
4	16	EID - The value of EID stored in the CF Environment of the originating device.
20	16	h(C), where C is a 64- or 128-bit control vector and h is a hash function. Basically, if C is 64 bits, then h(C) = concat(C,C). And, if C is 128 bits, then h(C) = C.
36	16	Key - This field contains a 128 bit odd parity adjusted key generated within the CF by the GKSP instruction.

CRYPTO FACILITY BACKUP DEA KEY RECORD (CFBDKR)

The Crypto Facility Backup DEA Key Record is produced by an ECFER instruction and is processed by an ICFER instruction. The CFBDKR is a 52 byte record.

A Crypto Facility Backup DEA Key Record has the following form:

Offset	Length	Data
(in bytes)		
0	1	Record ID The most significant bit in a byte is the leftmost bit. 0000 0001 - 'Crypto Facility Backup DEA Key Record'
1	1	Record Code PROTOCOL-MODE specified in ECFER (implying ICFER must also specify the same) xxxx 00xx - invalid xxxx 01xx - certification center protocol where the PUA control vector has HIST-CHAIN=2 xxxx 10xx - certification center protocol where the PUA control vector has HIST-CHAIN=3 xxxx 11xx - private protocol. KMP MODE specified in ECFER (implying ICFER must also specify the same) xxxx xx0x - KMP-mode = 0 has been specified in ECFER xxxx xx1x - KMP-mode = 1 has been specified in ECFER KM MODE specified in ECFER (implying ICFER must also specify the same) xxxx xxx0 - KM-mode = 0 has been specified in ECFER xxxx xxx1 - KM-mode = 1 has been specified in ECFER
2	1	Hash Rule Indicates the hash algorithm used to generate the hash. X'00': MDC-2 algorithm with 128-bit hash X'01': MDC-4 algorithm with 128-bit hash X'02': MD4 algorithm with 128-bit hash X'03' - X'FF' : reserved.
3	17	Reserved (=17 X'00')
20	16	MDC - A 128-bit MDC calculated on a CFER in the CF by an ECFER instruction. The MDC is calculated using the MDC-2 hash algorithm.
36	16	Key - This field contains a 128-bit odd parity adjusted key generated within the CF by the ECFER instruction, which may be Exclusive ORed with KM, KMP, or both (depending on KMP-mode and KM-mode specified in ECFER).

CRYPTO FACILITY SYSTEM SIGNATURE RECORD (CFSSR)

The Crypto Facility System Signature Record is produced by one of the following instructions: ECFAR, EPUK, GKSP, GDS, and ECFER. The CFSSR can be processed by one or more of the following instructions: IPUK, IDK, VDS, and ICFER. The CFSSR is a 253-bit record.

A Crypto Facility System Signature Record has the following form:

Offset	Length	Data
(in bits)		
0	4	Reserved (=B'0000')
4	1	Emulation B'1' : CFSSR created via the GDS instruction B'0' : CFSSR created via the instruction specified in the first nibble of Record Code field.
5	8	Record ID (=B'0000 0010' for CFSSR)
13	16	Record Length (in bits) The record length is currently fixed at

		253 bits (=X'00FD')
29	8	Record Code The first nibble indicates the CF instruction, and the second nibble indicates the key type of the private key used to generate the signature. First nibble: / Second nibble: B'0000' - ECFAR / B'1000' - PRC B'0001' - EPUK / B'1001' - PRM B'0010' - GKSP / B'1010' - PRA B'0011' - ECFER / B'1011' - PRU B'0100 - GDS / B'0xxx'. - reserved B'0101 - reserved / B'11xx' - reserved B'011x - reserved B'1xxx' - reserved
37	8	Hash Rule Indicates the hash algorithm used to generate the hash, and the rule (if any) for formatting and producing, from the generated hash value, the value to be stored in the Hash field. X'00': MDC-2 algorithm with 128-bit hash X'01': MDC-4 algorithm with 128-bit hash X'02': MD4 algorithm with 128-bit hash X'03' - X'FF' : reserved.
45	208	Hash field The field in which the hash value is stored-right justified, and filled with higher order zero bits.

CRYPTO FACILITY ENVIRONMENT RECORD (CFER)

The Crypto Facility Environment Record (CFER) contains that portion of a CF Environment necessary to "clone" a device (i.e., by replicating the CF Environment of one device into another device).

The Crypto Facility Environment Record has the following form:

Offset	Length	Data
(in bytes)		
00	64	Header (H)
64	a	Secret Part (SP)
64+a	b	Nonsecret Part (NSP)

DATA DESCRIPTION

HEADER The Header (H) contains information necessary to parse the CFER. H has a fixed length of 64 bytes.

SECRET PART The Secret Part (SP) contains the secret part of the CF Environment to be ported. SP is variable length, but a multiple of 8 bytes.

NONSECRET PART The Nonsecret Part (NSP) contains ONLY THAT PART OF THE NON- SECRET PART OF THE CF ENVIRONMENT to be ported. NSP is variable length, but contains a whole number of bytes.

The Header has the following form:

Offset	Length	Data
(in bytes)		
00	64	Header
00	01	Record ID (=B'00000011')

01	03	Reserved (=3 X'00')
04	04	Length of Secret Part in 8-byte blocks; (112+a+b)/8 Value is coded in binary representation.
08	04	Length of Secret Part of Product Environment in bytes ("a"). Value is coded in binary representation.
12	03	Reserved (=3 X'00')
15	01	Length of Random Pad for Secret Part in bytes ("b"). Value is coded in binary representation.
16	04	Length of Nonsecret Part in bytes; (568+d+e). Value is coded in binary representation.
20	02	Reserved (=2 X'00')
22	02	reserved (=2 X'00')
24	04	Length of Nonsecret Part of Product Environment in bytes ("e"). Value is coded in binary representation.
28	36	Reserved (=36 X'00')
64	00	End of Header

The Secret Part has the following form:

64	112+c	Secret Part
64	112	Registers
64	16	CKM Register (Current DEA-key-encrypting Master key)
80	16	OKM Register (Old DEA-key-encrypting Master key)
96	8	PRNGCTR1 Register (Pseudo-Random Number Counter #1)
104	8	PRNGCTR2 Register (Pseudo-Random Number Counter #2)
112	16	PRNGKEY1 Register (Pseudo-Random Number Seed Key #1)
128	16	PRNGKEY2 Register (Pseudo-Ransom Number Seed Key #2)
144	16	CKMP Register (Current PKA-key-encrypting Master key)
160	16	OKMP Register (Old PKA-key-encrypting Master key)
176	0	End of Registers
176	a	Secret Part of Product Environment The product environment contains information specific to a product implementation (beyond that called for by the PKCD).
176+a	b	Random Pad The Random Pad field contains "b" randomly generated

		pad bytes, where "b" is a number from 0 to 7. The random pad field is adjusted so that the length of the Secret Part is guaranteed to be a multiple of 8 bytes.
176+c	0	End of Secret Part (where c = a+b)

The Nonsecret Part has the following form:

176+c	488+f	Nonsecret Part
176+c	64	Configuration Vector
240+c	64	State Vector
		The following flags are reset to reflect that the corresponding registers do not port:
		KP FLAG := B'0'
		NKM FLAG := B'00'
		PIN FLAG := B'0'
		KMP FLAG(2) := B'00'
304+c	80	Registers
304+c	16	PROGMDC0 Register (Secure Loadable Program MDC #0)
320+c	16	PROGMDC1 Register (Secure Loadable Program MDC #1)
336+c	16	EID Register (Environment Identifier)
352+c	16	PUACV Register (Public Device Authentication key CV)
368+c	16	PRACV Register (Private Device Authentication key CV)
384+c	0	End of Registers
384+c	272	MDC Table
656+c	3	Counter Table
659+c	5	Reserved (=X'0000000000')
		Keeps remaining fields on an 8 byte boundary.
.* The next line is changed from PIN Tables to reserved		
664+c	d	reserved
664+c+d	e	Nonsecret Part of Product Environment
		The product environment contains information specific to a product implementation (beyond that called for by the PKCD).
664+c+f	0	End of Nonsecret Part (where f = d+e)
664+c+f	0	End of CFER

Outside the CF, the Secret Portion of the CFER is encrypted with a 128 bit DEA key KK1. KK1 is generated within the CF and encrypted with a public device authentication key PUA. The Nonsecret Portion of the CFER is specifically not encrypted to prevent a covert privacy channel from being set up when the CFER is used with the ECFER and ICFER instructions.

EXTERNAL CRYPTO FACILITY ENVIRONMENT RECORD (XCFER)

The External Crypto Facility Environment Record (XCFER) is the same as the CFER expect that the Secret Part is encrypted.

The External Crypto Facility Environment Record has the following form:

Offset	Length	Data
(in bytes)		
00	64	Header (H)
64	a	Encrypted Secret Part (ESP)
64 + a	b	Nonsecret Part (NSP)

DATA **DESCRIPTION**

HEADER The Header (H) contains information necessary to parse the CFER. H has a fixed length of 64 bytes.

ENCRYPTED SECRET PART The Encrypted Secret Part (ESP) contains the secret part of the CF Environment to be ported encrypted under a key shared with, or to-be-shared with, a designated receiving device. The length of ESP equals the length of SP. SP is variable length, but a multiple of 8 bytes.

NONSECRET PART The Nonsecret Part (NSP) contains the nonsecret part of the CF Environment to be ported. NSP is variable length, but a whole number of bytes.

CRYPTO FACILITY AUDIT RECORD (CFAR)

The Crypto Facility Audit Record (CFAR) contains the nonsecret part of the CF Environment plus additional nonsecret information. The CFAR is designed to be a multiple of 8 bytes.

The Crypto Facility Audit Record has the following form:

Offset	Length	Data
(in bytes)		
00	64	Header (H)
64	a	Nonsecret Part (NSP)

DATA DESCRIPTION

HEADER The Header (H) contains information necessary to parse the CFAR. It also contains a random number (RN) field and a date and time (DT) field. The Header is 64 bytes in length.

NONSECRET PART The Nonsecret Part (NSP) contains the nonsecret part of the CF Environment. NSP is variable length, but must be a whole number of bytes. The NSP in the CFAR is not the same as the NSP in the CFER (see Crypto Facility Environment Record).

The Header has the following form:

Offset	Length	Data
(in bytes)		
00	64	Header
00	01	Record ID (=B'00000100')
01	03	Reserved (=3 X'00')
04	04	Length of Nonsecret Part of CF Environment in bytes; (520+a+b+d). Value is coded in binary representation
08	02	Reserved (=2 X'00')
10	02	Length of cfpkr1 containing PUA ("a") in bytes. Value is coded in binary representation
12	02	Reserved (=2 X'00')
.* Next line is changed from PIN-table-length to reserved		
14	02	reserved (=2 X'00')
16	04	Length of Nonsecret Part of Product Environment ("d") in bytes. Value is coded in binary representation.
20	04	Reserved (=4 X'00')
24	08	RN field
32	03	Reserved (=3 X'00')
35	14	DT field
49	15	Reserved (=15 X'00')
64	00	End of Header

DATA DESCRIPTION

RN An 8 byte CFAP-supplied time-variant parameter. This field is set by the ECFAR instruction only when process-mode=1 or process-mode=2. This field is intended to be used as a nonce in a request/response protocol to guarantee freshness of the Audit record. The Certification Center generates and random number and sends it to the device to be audited in the Request-for-Audit message. The device then supplies this random number to the Export Cryptographic Facility Audit Record instruction. This results in the signed Audit record being sent to the Certification Center by the Audited device with the correct nonce. The Certification Center is assured that the Audit record is current.

DT A 14 character field with format YYYYMMDDHHMMSS containing the date and Coordinated Universal Time (UTC). The characters are decimal (0 thru 9) and are encoded using 8-bit ASCII representation. A value of 14 '0's denotes that DT is uninitialized.

The Nonsecret Part has the following form:

64	520+e	Nonsecret Part of CF Environment	
64	64	Configuration Vector	
128	64	State Vector	
192	112	Registers	
192	16	PROGMDC0 Register	
208	16	PROGMDC1 Register	
224	16	POS Register	
240	16	DID Register	
256	16	EID Register	
272	16	PUACV Register	
288	16	PRACV Register	
304	0	End of Registers	
304	272	MDC Table	
576	3	Counter Table	
579	5	Reserved (=5 X'00') Keeps remaining fields on an 8 byte boundary.	
584	a	cfpkrl from the PUA Buffer	
.* Next line is changed from PIN Tables to reserved.			
584+a	b	reserved	
584+c	0	GKSP Save	(not audited)
584+c	0	GKSP Buffer Length	(not audited)

584+c	0	GKSP Record Length	(not audited)
584+c	0	GKSP Buffer Flag	(not audited)
584+c	0	GKSP Ticket	(not audited)
584+c	0	IDK Save	(not audited)
584+c	0	IDK Buffer Length	(not audited)
584+c	0	IDK Record Length	(not audited)
584+c	0	IDK Buffer Flag	(not audited)
584+c	0	IDK Ticket	(not audited)
584+c	d	Nonsecret Part of Product Environment (where c = a+b) The product environment contains information specific to a product implementation (beyond that called for by the PKCD).	
584+e	0	End of Nonsecret Part of CF Environment (where e = c+d)	
584+e	0	End of CFAR	

No encrypted information in the CFER ever appears in the clear in the CFAR. Specifically, this is done to prevent a covert privacy channel from being set up when the CFER is used with the ECFER and ICFER instructions.

INTERNAL KEY UNIT (IKU)

The IKU is an internal form of a Key Unit. The Key Unit contains an encrypted CFPKR, an encrypted CFKAR, and information about the public or private key in the CFPKR. The IKU is designed to be a multiple of 8 bytes.

The Internal Key Unit has the following form:

offset	Length	Data
(in bytes)		
00	32	Header (H)
32	a	System Control Block (SCB)
32+a	b	User Control Block (UCB)
32+c	d	Encrypted Crypto Facility PKA Key Record (ECFPKR), c=a+b
32+e	f	Encrypted Crypto Facility Key Authenticator Record, e=c+d (ECFKAR)

DATA DESCRIPTION

HEADER The Header (H) contains information necessary to parse the IKU.

SYSTEM CONTROL BLOCK The System Control Block (SCB) contains information about the key in CFPKR, including a control vector C1. The SCB is managed by the system. The SCB is designed to be a multiple of 8 bytes.

USER CONTROL BLOCK The User Control Block (UCB) contains information about the key in CFPKR. The UCB is specified by the user (or installation). The UCB must be a multiple of 8 bytes.

ENCRYPTED CRYPTO FACILITY PKA KEY RECORD
The Encrypted Crypto Facility PKA Key Record (ECFPKR) contains a CFPKR encrypted under a key KMP.C2 formed as the Exclusive OR product of KMP and a control vector C2. C2 is generated from SCB and UCB using the method discussed in steps 501 and 502 of the Key Record Encrypt Algorithm 12 in Fig. 16. The CFPKR contains a public or private key.

ENCRYPTED CRYPTO FACILITY KEY AUTHENTICATOR RECORD
The Encrypted Crypto Facility Key Authenticator Record (ECFKAR) contains a CFKAR encrypted under a key KMP.C3 formed as the Exclusive OR product of KMP and a control vector C3. C3 is generated from SCB and UCB using the method described in steps 501 and 502 of the Key Record Encrypt Algorithm 12 in Fig. 16.

The Header has the following form:

Offset	Length	Data
(in bytes)		
00	32	Header (H)
00	02	Anti-ISO field (=X'8080') The anti-ISO field is a 2-byte field purposely encoded to be invalid as the leading 2 bytes of a data record conforming to 'Basic Encoding Rules of ASN.1(ISO 8825)'.
02	01	Record ID (=B'00000101')
03	03	(=3 X'00')
06	02	SCB-Length - number of 8 byte blocks in SCB. Value is coded in binary representation SCB-Length must be > 0
08	02	(=2 X'00')
10	02	UCB-Length - number of 8 byte blocks in UCB. Value is coded in binary representation UCB-Length must be >= 0
12	02	(=2 X'00')
14	02	ECFPKR-Length - number of 8 byte blocks in ECFPKR. Value is coded in binary representation ECFPKR-Length must be > 0
16	02	(=2 X'00')
18	02	ECFKAR-Length - number of 8 byte blocks in ECFKAR. Value is coded in binary representation ECFKAR-Length must be > 0
20	12	(=12 X'00')
32	0	End of Header (H)

The System Control Block has the following form:

Offset (in bytes)	Length	Data
32	a	System Control Block (SCB)
32	16	Control Vector
48	16	EID - Environment ID
64	2	Reserved (set to zero)
66	14	Tstart
80	2	Reserved (set to zero)
82	14	Texp
96	4	Reserved (set to zero)
100	4	Seq
104	64	LDID - Logical Device Identifier
168	64	LKN - Local Key Name
232	64	UID - User Identifier
296	b	Optional CFAP fields
296+b	0	End of System Control Block (SCB)

<u>DATA</u> <u>DESCRIPTION</u>

CONTROL VECTOR A 128 bit control vector associate with the public or private key stored in the CFPKR. The control vector is a CF enforced field. The control vector is a required field in the SCB.

EID A 16 byte Environment ID of the crypto facility where IKU is created. EID is a CF enforced field (i.e., the CF verifies that EID equals the value stored in the EID register of the CF when a key is created and, as appropriate, verifies that EID is equal or not equal to the value in the EID register when an IKU is processed). Note that EID may exist in multiple physical devices, depending on the number of "cloned" CF Environments active at any one time. EID is a required field in the SCB.

TSTART A 14 character field with format YYYYMMDDHHMMSS containing the date and Coordinated Universal Time (UTC) when the IKU becomes active. The characters in Tstart are decimal (0 thru 9) and are encoded using 8-bit ASCII representation. Tstart is a CF enforced field (i.e., the IKU cannot be processed unless Tstart has passed). A value of 14 ASCII '0's denotes that Tstart is ignored. Tstart is a required field in the SCB.

TEXP A 14 character field with format YYYYMMDDHHMMSS containing the date and Coordinated Universal Time (UTC) when the IKU expires. The characters in Texp are decimal (0 thru 9) and are encoded using 8-bit ASCII representation. Texp is a CF enforced field (i.e., the IKU cannot be processed when Texp has passed). A value of 14 ASCII '9's denotes that Texp is ignored. Texp is a required field in the SCB.

SEQ A 4 byte sequence number. Seq is not a CF enforced field. The seq field may be used by CFAP to record the relative sequence of IKU in a chain starting with a "root" IKU. Seq is an optional field in the SCB.

LDID Logical Device Identifier (LDID) is the identifier of the logical, as opposed to physical, device to which IKU belongs. LDID is not a CF enforced field. LDID consists of 1 or more name elements x_i separated by periods (i.e., x1, x2, x3 is stored as x1.x2.x3). Each name element x_i is 1 to 8 characters and is encoded in 8-bit ASCII representation. (Note that LDID is the network equivalent of EID.) LDID is an optional field in the SCB.

LKN Local Key Name (LKN) is the name or local name of the key in IKU and is assigned by the "logical" device to which IKU belongs. LKN is not a CF enforced field. LKN consists of 1 or more name elements x_i separated by periods (i.e., x1, x2, x3 is stored as x1.x2.x3). Each name element x_i is 1 to 8 characters and is encoded in 8-bit ASCII representation. LDID.LKN and UID.LKN constitute global key names that uniquely identify a key. LKN is an optional field in the SCB.

UID User Identifier (UID) is the identifier of the user to which IKU belongs. UID is not a CF enforced field. UID consists of 1 or more name elements x_i separated by periods (i.e., x1, x2, x3 is stored as x1.x2.x3). Each name element x_i is 1 to 8 characters and is encoded in 8-bit ASCII representation. UID is an optional field in the SCB.

CLEAR KEY UNIT (CKU)

The CKU is a clear form of an Internal Key Unit. The Key Unit contains a clear CFPKR and a clear CFKAR. The CKU is designed to be a multiple of 8 bytes.

The Clear Key Unit has the following form:

Offset	Length	Data
(in bytes)		
00	32	Header (H)
32	a	System control Block (SCB)
32+a	b	User Control Block (UCB)
32+c	d	Crypto Facility PKA Key Record (CFPKR), c=a+b
32+e	f	Crypto Facility Key Authenticator Record (CFKAR), e=c+d

DATA DESCRIPTION

HEADER The Header (H) contains information necessary to parse the CKU. See below.

SYSTEM CONTROL BLOCK

The System Control Block (SCB) contains information about the key in CFPKR, including a control vector C1. The SCB is managed by the system. The SCB is designed to be a multiple of 8 bytes. (The SCB form in the CKU is the same as in the IKU.)

USER CONTROL BLOCK

The User Control Block (UCB) contains information about the key in CFPKR. The UCB is specified by the user (or installation). The UCB must be a multiple of 8 bytes. (The UCB form in the CKU is the same as in the IKU.)

CRYPTO FACILITY PKA KEY RECORD

The Crypto Facility PKA Key Record (CFPKR) contains a public or private key.

CRYPTO FACILITY KEY AUTHENTICATOR RECORD

The Crypto Facility Key Authenticator Record (CFKAR) is used by the CF to validate the CFPKR.

The Header has the following form:

Offset	Length	Data
(in bytes)		
00	32	Header (H)
00	02	Anti-ISO field (=X'8080')

		The anti-ISO field is a 2-byte field purposely encoded to be invalid as the leading 2 bytes of a data record conforming to 'Basic Encoding Rules of ASN.1(ISO 8825)'.
02	01	Record ID (=B'00000110')
03	03	Reserved (=3 X'00')
06	02	SCB-Length - number of 8 byte blocks in SCB. Value is coded in binary representation. SCB-Length must be > 0
08	02	Reserved (=2 X'00')
10	02	UCB-Length - number of 8 byte blocks in UCB. Value is coded in binary representation. UCB-Length must be >= 0
12	02	Reserved (=2 X'00')
14	02	CFPKR-Length - number of 8 byte blocks in CFPKR. Value is coded in binary representation CFPKR-Length must be > 0
16	02	reserved (=2 X'00')
18	02	CFKAR-Length - number of 8 byte blocks in CFKAR. Value is coded in binary representation. CFKAR-Length must be > 0
20	12	Reserved (=12 X'00')
32	0	End of Header (H)

NOTE: The specification for System Control Block, User Control Block, Crypto Facility PKA Key Record, and Crypto Facility Key Authenticator Record are the same as those for the IKU.

EXTERNAL KEY UNIT (EKU)

The EKU is an external form of a Key Unit. The Key Unit contains a clear CFPKR and information about the public or private key in the CFPKR. The EKU has no encrypted or clear CFKAR. The EKU is designed to be a multiple of 8 bytes.

The External Key Unit has the following form:

Offset	Length	Data
(in bytes)		
00	32	Header (H)
32	a	System Control Block (SCB)
32 + a	b	User Control Block (UCB)
32 + c	d	Crypto Facility PKA Key Record (CFPKR), c = a + b

DATA DESCRIPTION

HEADER The Header (H) contains information necessary to parse the EKU.

SYSTEM CONTROL BLOCK

The System Control Block (SCB) contains information about the key in CFPKR, including a control vector C1. The SCB is managed by the system. The SCB is designed to be a multiple of 8 bytes. (The SCB form in the EKU is the same as in the IKU.)

USER CONTROL BLOCK

The User Control Block (UCB) contains information about the key in CFPKR. The UCB is specified by the user (or installation). The UCB must be a multiple of 8 bytes. (The UCB form in the EKU is the same as in the IKU.)

CRYPTO FACILITY PKA KEY RECORD

The Crypto Facility PKA Key Record (CFPKR) contains a public or private key, although ordinarily only public keys occur in an EKU.

The Header has the following form:

Offset	Length	Data
(in bytes)		
00	32	Header (H)
00	02	Anti-ISO field (=X'8080') The anti-ISO field is a 2-byte field purposely encoded to be invalid as the leading 2 bytes of a data record conforming to 'Basic Encoding Rules of ASN.1(ISO 8825)'.
02	01	Record ID (=B'00000111')
03	03	Reserved (=3 X'00')
06	02	SCB-Length - number of 8 byte blocks in SCB. Value is coded in binary representation. SCB-Length must be > 0
08	02	Reserved (=2 X'00')
10	02	UCB-Length - number of 8 byte blocks in UCB. Value is coded in binary representation. UCB-Length must be >= 0
12	02	Reserved (=2 X'00')
14	02	CFPKR-Length - number of 8 byte blocks in CFPKR. Value is coded in binary representation. CFPKR-Length must be > 0
16	02	reserved (=2 X'00')
18	02	Constant (=2 X'00')
20	12	(=12 X'00')
32	0	End of Header (H)

NOTE: The specification for System Control Block, User Control Block, and Crypto Facility PKA Key Record Record are the same as those for the IKU.

SKELETON KEY UNIT (SKU)

The SKU is a partially completed Key Unit. The SKU is designed to be a multiple of 8 bytes. The Skeleton Key Unit has the following form:

Offset	Length	Data
(in bytes)		
00	32	Header (H)
32	a	System Control Block (SCB)
32 + a	b	User Control Block (UCB)

DATA	DESCRIPTION
HEADER	The Header (H) contains information necessary to parse the SKU.
SYSTEM CONTROL BLOCK	The System Control Block (SCB) contains information about the key in CFPKR, including a control vector C1. The SCB is managed by the system. The SCB is designed to be a multiple of 8 bytes. The SCB format is the same as that for the IKU.
USER CONTROL BLOCK	The User Control Block (UCB) contains information about the key in CFPKR. The UCB is specified by the user (or installation). The UCB is an optional field in the SKU. The UCB must be a multiple of 8 bytes.

The Header has the following form:

Offset	Length (in bytes)	Data
00	32	Header (H)
00	02	(=X'8080')
02	04	(=4 X'00')

06	02	SCB-Length - number of 8 byte blocks in SCB. Value is coded in binary representation. SCB-Length must be > 0
08	02	(=2 X'00')
10	02	UCB-Length - number of 8 byte blocks in UCB. Value is coded in binary representation. UCB-Length must be >= 0
12	02	(=2 X'00')
14	02	Constant (=2 X'00')
16	02	(=2 X'00')
18	02	Constant (=2 X'00')
20	12	(=12 X'00')
32	0	End of Header (H)

NOTE: The specification for System Control Block and User Control Block are the same as those for the IKU.

CONTROL VECTOR FORMATS AND DESCRIPTIONS

AN OVERVIEW OF PKCD KEY TYPES

Fig. 30 illustrates the PKCD control vector hierarchy. Each PKCD control vector has a CV TYPE field consisting of a main-type and a sub-type. The main-type portion of the CV TYPE field permits broad classes of keys and cryptovariables to be defined, whereas the sub-type portion of the CV TYPE field permits generic key types to be defined within each class, which are more closely associated with the functional use of the key or cryptovariable. The lefthand portion of Fig. 30 illustrates the control vector main-types defined by PKCD. The righthand portion of Fig. 30 illustrates the control vector sub-types defined for each main-type. When no sub-type distinction is made, the key or cryptovariable is generally referred to by its main-type.

The PKCD names ascribed to keys are determined by a concatenation of the names associated with main-type and sub-type. The following keys are defined by PKCD:

- o Public Authentication Key
- o Public Certification Key
- o Public Key Management Key
- o Public User Key
- o Private Authentication Key
- o Private Certification Key
- o Private Key Management Key
- o Private User Key

GENERAL FORMAT FOR PKA CONTROL VECTORS

The fields defined for one or more control vectors are these:

ALGORITHM	GKSP	LENGTH
ALGORITHM EXTENSION	HIST-CHAIN	PARITY
ANTIVARIANT ONE	HIST-DOMAIN ID	PR USAGE
ANTIVARIANT ZERO	HIST-IPRK	PU USAGE
CV TYPE	HIST-IPUK	RTNKMP/RTCKMP
DOMAIN ID	HIST-KREGMODE	SOFTWARE
ECFAR	HIST-MDC	TESTZERO
ECFER	ICFER	THRES-MDC
EPUK	IDK	VAL/VAL AUTHENTICATOR
EXTENSION	INSTALLATION	
GADS	IPUK	
GDS	KREGMODE	

A definition of the control vector fields is provided below in alphabetical order:

ALGORITHM (4 BITS)

This field contains an algorithm unique code word which permits the CF to distinguish keys associated with one PKA from another. (The architecture permits the CF to implement multiple PKAs.) Each different PKA is assigned a different code word. The ALGORITHM field is checked before a key is used by the PKA, thus preventing keys associated with one PKA to be used with another PKA. The coding of this field is as follows:

o B'0000' : RSA Algorithm (modulus size from 512 to 2048 bits)
o B'0001'-B'1111' : reserved

ALGORITHM EXTENSION (3 BITS)

This field is an extension of the ALGORITHM field, and the coding is dependent on the value of the ALGORITHM field.
For ALGORITHM field = B'0000', the coding of the ALGORITHM EXTENSION field is as follows:

o B'000' : No restrictions
o B'001' : Public key exponent is 3
o B'010' : Public key exponent is 2¯16 + 1
o B'011'-B'111' : reserved

ANTIVARIANT ONE (1 BIT)

This field is a fixed value of B'1'.

ANTIVARIANT ZERO (1 BIT)

This field is a fixed value of B'0'.

CV TYPE (7 BITS)

This field indicates the type of the control vector, which is also the key type of the key with which this control vector is associated. The following key types are defined for PKA keys :

1. B'1110010': Public Authentication Key
2. B'1111010': Private Authentication Key
3. B'1110000': Public Certification Key
4. B'1111000': Private Certification Key
5. B'1110001': Public Key Management Key
6. B'1111001': Private Key Management Key
7. B'1110011': Public User Key
8. B'1111011': Private User Key

Note that the value of the first three bits of the CV TYPE field of PKA control vectors are always B'111', as opposed to other values for DEA control vectors.

DOMAIN ID (4 BITS)

This field contains a domain identifier (an arbitrary value from B'0000' to B'1111' assigned by an installation). The domain ID field of all public and private keys used within a cryptographic instruction must be equal.

ECFAR (1 BIT)

This field indicates whether a private key PR can be used in an ECFAR instruction to generate a digital signature on a CFAR:

o B'0' : cannot
o B'1' : can

ECFER (1 BIT)

In a PRA control vector, this field indicates whether a PRA key can be used in the ECFER instruction to generate a digital signature on an XCFER. In a PUA control vector, this field indicates whether a PUA key can be used to encrypt a CFBDKB.

o B'0' : cannot
o B'1' : can

EPUK (1 BIT)

This field indicates whether a private key can be used in an EPUK instruction to generate a digital signature on an output External Key Unit (EKU).

o B'0' : cannot
o B'1' : can

EXTENSION (2 BITS)

This field indicates whether the control vector is a 64-bit, 128-bit, or >128-bit control vector. In PKCD , all control vectors are >128-bit control vectors.

o B'00' : 64 bit control vector base
o B'01' : the control vector is a 128-bit control vector
o B'10' : the control vector is a >128-bit control vector
o B'11' : reserved

GADS (1 BIT)

This field indicates whether a private key (PRC, PRM or PRU) can be used in a GADS instruction to generate a digital signature.

o B'0' : cannot
o B'1' : can

GDS (1 BIT)

This field indicates whether a private key (PRA, PRC, PRM or PRU) can be used in a GDS instruction to generate a digital signature.

o B' 0' : cannot

o B'1' : can

GKSP (1 BIT)

This field indicates whether a key (PRM or PUM) can be used in a GKSP instruction.

o B'0' : cannot

o B'1' : can

HIST-CHAIN (2 BITS)

This field indicates a chain of history of how a public key has been imported in the IPUK instruction:

o B'00' : other (i.e., not B'01', B'10', or B'11')

o B'01' : conditions stated in (a) or (b) must be true:

- (a) PU in EKU1 is a PUC and is imported via import-mode = 0;
- (b) PU in EKU1 is a PUC and is imported via import-mode = 1; PU in IKU2 is a PUC with HIST-IPUK = 1 and HIST-CHAIN = 1; PU in EKU1 and PU in IKU2 have same DOMAIN ID.

o B'10' : conditions stated in (c) or (d) must be true:

- (c) PU in EKU1 is a PUM and is imported via import-mode = 1; PU in IKU2 is a PUC with HIST-IPUK = 1 and HIST-CHAIN = 1; PU in EKU1 and PU in IKU1 have same DOMAIN ID.
- (d) PU in EKU1 is a PUA and is imported via import-mode = 1; PU in IKU2 is a PUC with HIST-IPUK = 1 and HIST-CHAIN = 1.

o B'11' : conditions stated in (e) must be true:

- (e) : PU in EKU1 is a PUA with HIST-IPUK = 0 and is imported via import-mode = 1; PU in IKU2 is a PUM with HIST-IPUK = 1 and HIST-CHAIN = 2.

NOTE: this field is valid only when HIST-IPUK = B'1'.

HIST-DOMAIN ID (4 BITS)

HIST-DOMAIN ID is a field in a PUA control vector used to record the value of DOMAIN ID in a PUC or PUM control vector. A domain identifier is an arbitrary value from B'0000' to B'1111'. PUA is a key in an EKU imported with IPUK and PUM or PUC is a key used to validate the digital signature previously generated on the to-be-imported EKU at the sending device.

NOTE: this field is valid only when HIST-IPUK = B'1' and either HIST-CHAIN = B'10' or HIST-CHAIN = B'11'.

HIST-IPRK (1 BITS)

This field indicates whether a private user key has been imported via the IPRK instruction, as follows:

o B'0' : not imported via IPRK

o B'1' : imported via IPRK

HIST-IPUK (1 BITS)

This field indicates whether a public key (PUA, PUC, PUM, or PUU) has been imported via the IPUK instruction, as follows:

o B'0' : not imported via IPUK

o B'1' : imported via IPUK

NOTE: the HIST-MDC and HIST-CHAIN fields in the control vector are valid only when HIST-IPUK in the control vector = B'1'. HIST-KREGMODE is valid only when HIST-IPUK = B'1' and HIST-CHAIN = B'11'

HIST-KREGMODE (2 BITS)

HIST-KREGMODE is a field in a PUA control vector used to record the value of KREGMODE in a PUM control vector. See also KREGMODE. PUA is a key in an EKU imported with IPUK and PUM is a key used to validate the digital signature previously generated on the to-be-imported EKU at the sending device.

o B'00' : KREGMODE = B'00' in PUM
o B'01' : KREGMODE = B'01' in PUM
o B'10' : KREGMODE = B'10' in PUM
o B'11' : reserved

NOTE: this field is valid only when HIST-IPUK = B'1' and HIST-CHAIN = B'11'.

HIST-MDC (2 BITS)

This field records IPUK information about a root PU in a chain, as follows:

o B'00' : reserved
o B'01' : root PU was imported in IPUK using MDC-mode = 0 (i.e., no MDC)
o B'10' : root PU was imported in IPUK using MDC-mode = 1 (i.e, with MDC) such that EKUMDC FLAG = B'10'.
o B'11' : root PU was imported in IPUK using MDC-mode = 1 (i.e., with MDC) such that EKUMDC FLAG = B'11'.

NOTE: this field is valid only when HIST-IPUK = B'1'.

ICFER (1 BIT)

In a PUA control vector, this field indicates whether a PUA key can be used in the ICFER instruction to validate a digital signature on an XCFER. In a PRA control vector, this field indicates whether a PRA key can be used to decrypt an encrypted CFBDKB.

o B'0' : cannot
o B'1' : can

IDK (1 BIT)

This field indicates whether a key (PRM or PUM) can be used in an IDK instruction.

o B'0' : cannot
o B'1' : can

INSTALLATION (7 BITS)

This field represents control vector bits that are controlled/managed entirely by the installation. The INSTALLATION field is not checked/enforced by the hardware (CF).

IPUK (1 BIT)

This field indicates whether a public key can be used in an IPUK instruction to validate a digital signature on an input External Key Unit (EKU).

o B'0' : cannot
o B' 1' : can

NOTE: the IPUK usage bit does not control the use of PU in an EKU to validate a signature on that same EKU.

KREGMODE (2 BITS)

This field indicates the method used to register a public key management key (PUM) in a certification center environment.

o B'00' : PUM not registered
o B'01' : PUM registered without restrictions
o B'10' : PUM registered with restrictions
o B'11' : reserved

LENGTH (16 BITS)

This field contains a length value which directly or indirectly determines key length or key size. The coding and interpretation of the LENGTH field is dependent of the ALGORITHM field.

For ALGORITHM = B'0000' (i.e., RSA) the LENGTH field contains a value from 512 to 2048 representing modulus length in bits. Unless elsewhere restricted, the public and private keys can range in length up to the modulus length. The key generator shall ensure that if LENGTH = n, then a modulus is generated such that the value of the modulus is B'1' followed by n-1 zero and one bits.

PARITY (16 BITS)

This is a set of bits in the control vector reserved for use by CFAP and by the the algorithm used to calculate the Hash Function h. The PARITY bits are used to set even byte parity on the 128-bit value H=h-(C) produced by applying Hash Function h to control vector C.

PR USAGE (7 BITS)

In a PR control vector, PR USAGE consists of architected usage bits and reserved bits. The PR USAGE field is also stored as history information in the associated PU control vector.

The following PR USAGE subfields are defined for a PRA control vector:

o ECFAR (1 bit)
o EPUK (1 bit)
o ECFER (1 bit)
o ICFER (1 bit)
o GDS (1 bit)

The following PR USAGE subfields are defined for a PRC control vector:

o ECFAR (1 bit)
o RTNPMK/RTCPMK (1 bit), reserved (=B'1')
o EPUK (1 bit)
o GDS (1 bit)
o GADS (1 bit)

The following PR USAGE subfields are defined for a PRM control vector:

o ECFAR (1 bit)
o RTNPMK/RTCPMK (1 bit), reserved (=B'1')
o EPUK (1 bit)
o GDS (1 bit)
o GKSP (1 bit)
o IDK (1 bit)
o GADS (1 bit)

The following PR USAGE subfields are defined for a PRU control vector:

o ECFAR (1 bit)
o RTNPMK/RTCPMK (1 bit), reserved (=B'1')
o EPUK (1 bit)
o GDS (1 bit)
o GADS (1 bit)

PU USAGE (7 BITS)

In a PU control vector, PU USAGE consists of architected usage bits and reserved bits. The PU USAGE field is also stored as history information in the associated PR control vector.

The following PU USAGE subfields are defined for a PUA control vector:

o RTNPMK/RTCPMK (1 bit), reserved (=B'1')
o IPUK (1 bit)
o ECFER (1 bit)
o ICFER (1 bit)

The following PU USAGE subfields are defined for a PUC control vector:

o RTNPMK/RTCPMK (1 bit), reserved (=B'1')
o IPUK (1 bit)

The following PU USAGE subfields are defined for a PUM control vector:
o RTNPMK/RTCPMK (1 bit), reserved (= B'1')
o IPUK (1 bit)
o GKSP (1 bit)
o IDK (1 bit)
The following PU USAGE subfields are defined for a PUU control vector:
o RTNPMK/RTCPMK (1 bit), reserved (= B'1')
o IPUK (1 bit)

RTNKMP/RTCKMP (1 BIT)

This field indicates whether a public or private key can be reenciphered in an RTNKMP or RTCKMP instruction:
o B' 0' : cannot
o B'1' : can
NOTE: This field has a fixed value of B'1', and is enforced in the GDAK and GPUPR instruction.

SOFTWARE (6 BITS)

This field represents control vector bits that are controlled/managed entirely by CFAP. The SOFTWARE field is not checked/enforced by the hardware (CF).

TESTZERO (3 BITS)

This field is reserved by the CF and tested for zero. That is, TESTZERO must equal B'000'.

THRES-MDC (2 BITS)

This field is used in a PRMa control vector to establish a threshold on HIST-MDC in a corresponding PUMb control vector. The PRMa and PUMb are used together in a GKSP or IDK instruction. Note that "a" represents this device and "b" another device.
o B'00' : reserved
o B'01' : HIST-MDC must be > = B'01'
o B'10' : HIST-MDC must be > = B'10'
o B'11' : HIST-MDC must be = B'11'

VALUE/AUTHENTICATOR (1 BIT)

The VALUE/AUTHENTICATOR field is reserved for use by the algorithm used to calculate the Hash Function h.
The layout of control vectors for all PKCD keys are described in Figs. 31 through 38, inclusive.

GENERAL FORMAT FOR THE HASH VECTOR

A definition of the hash vector fields is provided below in alphabetical order:

ANTIVARIANT ONE (1 BIT)

This field is a fixed value of B'1'.

ANTIVARIANT ZERO (1 BIT)

This field is a fixed value of B'0'.

EXTENSION (2 BITS)

This field indicates whether the hash vector is produced from a 64-bit, 128-bit, or >128-bit control vector. In PKCD , all hash vectors are produced from >128-bit control vectors.

- o B'00' : hash vector produced from 64 bit control vector
- o B'01' : hash vector produced from 128-bit control vector
- o B'10' : hash vector produced from >128-bit control vector
- o B'11' : reserved

HASH (107 BITS)

The HASH field Consists of 107 bits of a 128 bit Modification Detection Code (MDC) produced by using the MDC-2 hash algorithm. The HASH field consists of bits 00. .06, 08. .14, 16. .22, 24. .29, 32. .37, 40. .44, 48. .54, 56. .61, 64. .70, 72. .78, 80. .86, 88. .94, 96. .102, 104. .110, 112. .118, 120. .126 from the MDC.

PARITY (16 BITS)

The PARITY bits are used to set even byte parity on the 128 hash vector.

VALUE/AUTHENTICATOR (1 BIT)

The VALUE/AUTHENTICATOR field indicates whether the hash vector is associated with a value or an authenticator, as follows:

- o B'0' : value
- o B'1' : authenticator

The layout of the Hash vector is described in Fig. 39.

INSTRUCTION PROCESSING

INSTRUCTION SET

The CF instructions may be logically divided into eight functional categories:

CF INITIALIZATION

These instructions support various CF initialization, including the PKA master key.

CF CONFIGURATION

This instruction is used to load a configuration vector into the CF.

CF AUDIT

This instruction is used to export the nonsecret portion of the CF environment.

CF CONTROL

These instructions are used to control instruction execution and to change CF state.

CKDS UPDATE

These instructions are used to reencipher the keys in a CKDS from a current to a new, or an old to a current, PKA master key.

KEY MANAGEMENT

These instructions are used to generate, export, and import PKCD PKA keys. They are also used to generate and import DEA key-encrypting keys.

SYSTEM DIGITAL SIGNATURES

These instructions are used to generate and verify system digital signatures.

APPLICATION DIGITAL SIGNATURES

These instructions are used to generate and verify application digital signatures.

CRYPTO FACILITY BACKUP

These instructions are used to export and import a CF environment.

UTILITY

These instructions provide miscellaneous cryptographic functions.

The instructions are listed by group in the following table:

Table 1 (Page 1 of 3).

INSTRUCTION NAME	INSTRUCTION MNEMONIC
CF INITIALIZATION:	
Load Physical Identifier	LPID
Generate Device Authentication Key Pair	GDAK
Load First PKA Master Key Part	LFPMKP
Combine PKA Master Key Parts	CPMKP

Table 1 (Page 2 of 3).

INSTRUCTION NAME	INSTRUCTION MNEMONIC
Generate New PKA Master Key	GNPMK
Generate New DEA Master Key	GNDMK
Set PKA Master Key	SPMK
Load MDC For Public Certification Key	LMDCC
Load MDC	LMDC
Initialize Pseudo Random Number Generator	IPRNG
CF CONFIGURATION:	
Load Configuration Vector	LCV
CF AUDIT:	
Export Crypto Facility Audit Record	ECFAR
CF CONTROL:	
Enter Preinit State	EPS
Enter Init State	EIS
Enter Run State	ERS
Clear New PKA Master Key Register	CLNPMK
Clear Old PKA Master Key Register	CLOPMK
Set Authorization Flag	SAF
Set Enable Flag	SEF
CKDS UPDATE:	
Reencipher to New PKA Master Key	RTNPMK
Reencipher to Current PKA Master Key	RTCPMK

Table 1 (Page 3 of 3).

INSTRUCTION NAME	INSTRUCTION MNEMONIC
KEY MANAGEMENT:	
Generate Public and Private Key Pair	GPUPR
Export Public Key	EPUK
Import Public Key	IPUK
Import Private Key	IPRK
Generate Key Set PKA	GKSP
Import DEA Key	IDK
Verify Internal Key Unit	VIKU
SYSTEM DIGITAL SIGNATURES:	
Generate Digital Signature	GDS
Verify Digital Signature	VDS
APPLICATION DIGITAL SIGNATURES:	
Generate Application Digital Signature	GADS
Verify Application Digital Signature	VADS
CRYPTO FACILITY BACKUP:	
Export Crypto Facility Environment Record	ECFER
Import Crypto Facility Environment Record	ICFER
UTILITY:	
Set and Reset Alarm	SRALM

CF INITIALIZATION

Load Physical Device ID (LPID)

EQUATION:

```
          PID                 /128b/
          -->
          CC                  /unspecified/
```

PARAMETER DEFINITIONS:

INPUTS	DESCRIPTION
PID	A 128 bit physical identifier of a device.

OUTPUTS	DESCRIPTION
CC	Condition code indicating success or failure of the instruction execution.

DESCRIPTION:

The Load Physical Device ID instruction permits a 128 bit physical identifier of a device to be loaded into the CF and stored in the DID and EID registers. Execution of the LPID instruction causes the DID flag to be set to the "full" state. The instruction executes only when the DID flag is in the "empty" state. (Note that an EPS instruction must be executed in order to reset the DID flag to the "empty" state.) The DID flag serves two purposes: (a) it controls the execution of the LPID instruction, and (b) it indicates whether the DID and EID registers have or have not been initialized.

The value of PID stored in the DID register is the PID value associated with PUA and PRA (i.e., the PUA and PRA of that device).

The value of PID stored in the EID register is used for two purposes: (a) it is the value stored in the EIDO field of a certificate, and thus identifies the device to another device, and (b) it is the value stored in a DEA key record, which is used by the GKSP and IDK instructions as an antireimport value.

The 16 byte PID consists of an 8 byte network part and an 8 byte node part. The 8 byte node part uniquely identifies the node within a network. The 8 byte network part uniquely identifies the network. The objective is to arrive at a naming convention that will ensure unique PID values from one network to another. One possibility is for the 8 byte network part to be registered (e.g., with an IBM registration center).

The ECFAR instruction can be used by CFAP to read the contents of the DID and EID registers.

For reasons of security, the LPID instruction is architected such that the DID register contents cannot be changed without erasing the contents of the PUA and PRA buffers (i.e., a different PID can't be assigned to the same key pair stored in the PUA and PRA buffers). In like manner, the ICFER instruction is architected such that the EID register contents cannot be changed without reinitializing the CKMP register with a new key. Otherwise, use of the EID buffer as an anti-reimport value would be ineffective.

The LPID instruction executes only in the "preinit" state.

Generate Device Authentication Key Pair (GDAK)

EQUATION:

```
C1          /128 bits/
C2          /128 bits/
-->
CC          /unspecified/
```

PARAMETER DEFINITIONS:

INPUTS	DESCRIPTION
C1	A 128 bit control vector associated with the generated public authentication key PUA.
C2	A 128 bit control vector associated with the generated private authentication key PRA.

OUTPUTS	DESCRIPTION
CC	Condition code indicating success or failure of the instruction execution.

DESCRIPTION:

The Generate Device Authentication Key Pair instruction generates a public and private authentication key pair, PUA and PRA. The generated keys are stored in the PUA buffer and PRA buffer in the CF, respectively, as Crypto Facility PKA Key Record 1 (CFPKR1) and Crypto Facility PKA Key Record 2 (CFPKR2). The 128 bit control vectors associated with PUA and PRA are specified to the GDAK instruction as inputs C1 and C2, respectively. The control vectors specify the public key algorithm and other algorithm related information necessary for key generation. Consistency checking is performed on C1 and C2. For example, the ALGORITHM, ALGORITHM EXTENSION, and LENGTH fields in C1 and C2 must match.

Execution of the GDAK instruction causes the GDAK FLAG in the state vector to be set to the "full" state from the "empty" state. The instruction executes only when the GDAK FLAG is in the "empty" state. (Note that the EPS instruction must be executed to reset the GDAK FLAG to the "empty" state.)

The GDAK FLAG serves two purposes: (a) it controls execution of the GDAK instruction, and (b) it indicates when the PUA and PRA buffers have been initialized.

The GDAK instruction executes only in the "preinit" state.

FUNCTIONAL SPECIFICATION:

1. Perform input parameter consistency checking: None.
2. Perform state vector checking:
   a. Verify that CF STATE in the state vector is in the "preinit" state.
   b. Verify that GDAK FLAG in state vector is in the "empty" state.

Continue if checking succeeds: otherwise set CC status flag and jump to step 8.

3. Perform control vector checking. Continue if checking succeeds: otherwise set CC status flag and jump to step 8.
4. Store control vectors:
   a. Store C1 in PUACV Register
   b. Store C2 in PRACV Register
5. Generate a pair of cryptographic facility PKA records cfpkr1 and cfpkr2 of length s1 and s2, respectively, where s1 and s2 are pre-selected values that indicate the number of 8 byte blocks.
6. Store generated keys and lengths:
   a. Store s1 in PUA Buffer Length field in CF Environment.
   b. Store cfpkr1 in PUA Buffer in CF Environment.
   c. Store s2 in PRA Buffer Length field in CF Environment.
   d. Store cfpkr2 in PRA Buffer in CF Environment.
7. Perform state vector update.
   a. Set GDAK FLAG to the "full" state.
8. Produce output CC from CC status flags.

CONTROL BLOCK AND CONTROL VECTOR CHECKING:

Perform control vector checking:

1. Checking on C1 (associated with PUA)
   a. Verify CV TYPE = 'public authentication key'
   b. Note: checking on CV TYPE EXTENSION has been deleted.
   c. Verify RTNKMP/RTCKMP usage bit = B'1'
   d. Perform Control Vector Validate on C1 to validate certain fields in C1.
   e. Verify RC1 = 0.

   If any of the above checking fails then stop the control vector checking and issue a condition code to indicate that C1 is not valid.
2. Checking on C2 (associated with PRA):
   a. Verify CV TYPE = 'private authentication key'
   b. Perform Control Vector Validate on C1 to validate certain fields in C1.
   c. Verify RC1 = 0.

   If any of the above checking fails then stop the control vector checking and issue a condition code to indicate that C2 is not valid.
3. Checking on C1 and C2:
   a. Note: checking on CV TYPE EXTENSION has been deleted.
   b. Verify ALGORITHM in C1 = ALGORITHM in C2
   c. Verify ALGORITHM EXTENSION in C1 = ALGORITHM EXTENSION in C2
   d. Verify LENGTH in C1 = LENGTH in C2
   e. Verify PR USAGE in C1 = PR USAGE in C2
   f. Verify PU USAGE in C1 = PU USAGE in C2

   If any of the above checking fails then stop the control vector checking and issue a condition code to indicate that cross checking among control vectors has failed.

   Note that there is no cross checking on (1) DOMAIN ID since this field is not implemented in the PUA and PRA control vectors.

Load First PKA Master Key Port (LFPMKP)

EQUATION:

```
input-mode          /1b minimum/
<key-part>          /128b/              ; if input-mode=0
-->
CC                  /unspecified/
```

PARAMETER DEFINITIONS:

INPUTS	DESCRIPTION
INPUT-MODE	specifies how the key part to be processed is supplied to the instruction. o 0 : the key part is passed via the instruction interface, i.e., via input parameter key-part. o 1 : the key part is retrieved from the internal KP register.
KEY-PART	128 bit key part. This parameter is required only when input-mode=0.

OUTPUTS	DESCRIPTION
CC	Condition code indicating success or failure of the instruction execution.

DESCRIPTION:

The Load First PKA Master Key Part instruction loads the first part of the PKA master key (KMP) into the NKMP (New PKA Master Key) register. An input-mode parameter indicates whether the loaded key part is passed as an input parameter at the instruction interface or whether it is retrieved from the internal KP register. The NKMP flag is set to the "partially full" state from the "empty" state and the NKMP History Flag is set to 0 (indicating that the contents of the NKMP register were loaded via the LFPMKP instruction). If input-mode = 1, the operation is performed only if the KP flag is in the "full" state; in which case the KP flag is set to the "empty" state. The operation is performed only if the NKMP flag is in the "empty" state.
NOTE: If input-mode = 1, it is assumed that prior to the execution of this instruction the first PKA master key part has been entered into the key part register via a key-entry device, keyboard, etc., which, optionally, may operate only in a special authorized mode (e.g., supersecure mode enabled via a physical key-activated switch).

The LFPMKP instruction executes only in the "run" state.

Combine PKA Master Key Parts (CPMKP)

EQUATION:

```
input-mode          /1b minimum/
mode                /1b minimum/
<key-part>          /128b/          ; if input-mode=0
-->
CC                  /unspecified/
```

PARAMETER DEFINITIONS:

INPUTS	DESCRIPTION
INPUT-MODE	specifies how the key part to be processed is supplied to the instruction.
	o 0 : the key part is passed via the instruction interface, i.e., via input

parameter key-part.

- o 1 : the key part is retrieved from the internal KP register.

MODE	indicates whether the PKA master key part in the key part register is an intermediate key part or a last key part. o 0 : intermediate key part o 1 : last key part
KEY-PART	128 bit key part. This parameter is required only when input-mode=0.
OUTPUTS	**DESCRIPTION**
CC	Condition code indicating success or failure of the instruction execution.

DESCRIPTION:

The Combine PKA Master Key Parts instruction Exclusive ORs a PKA master key part with the PKA master key part stored in the NKMP register and stores the result in the NKMP register. An input-mode parameter indicates whether the loaded key part is passed as an input parameter at the instruction interface or whether it is retrieved from the internal KP register. The NKMP flag is set to the "full" state if mode = 1 or to the "partially full" state if mode = 0. For mode=1, the CPMKP instruction ensures that the produced value of KMP has odd parity (odd parity adjusted, if necessary) and that the left and right 64 bit parts of KMP are not equal. If input-mode=1, the operation is performed only if the KP flag is in the "full" state; in which case the KP flag is set to the "empty" state. The operation is performed only if the NKMP flag is in the "partially full" state and the NKMP History flag is zero. The instruction has no output.

NOTE: If input-mode=1, it is assumed that prior to the execution of this instruction a PKA master key part has been entered into the key part register via a key-entry device, keyboard, etc., which, optionally, may operate only in a special authorized mode (e.g., supersecure mode enabled via a physical key-activated switch).

The CPMKP instruction executes only in the "run" state.

Generate New PKA Master Key (GNPMK)

EQUATION:

```
()
-->
CC                     /unspecified/
```

PARAMETER DEFINITIONS:

INPUTS	DESCRIPTION
	None.

OUTPUTS	DESCRIPTION
CC	Condition code indicating success or failure of the instruction execution.

DESCRIPTION:

The Generate New PKA Master Key instruction causes a 128 bit odd parity adjusted random number to be generated and stored in the NKMP register. The left and right 64 bit parts of the generated key must be unequal. The instruction executes only if the NKMP flag is in the "empty" state. Successful execution Of the GNPMK instruction causes the NKMP flag to be set to the "full" state from the "empty" state and the NKMP History flag to be set := B'1'.

The GNPMK instruction executes only in the "run" state.

Generate New DEA Master Key (GNDMK)

EQUATION:

```
()
-->
CC                    /unspecified/
```

PARAMETER DEFINITIONS:

INPUTS	DESCRIPTION
	None.

OUTPUTS	DESCRIPTION
CC	Condition code indicating success or failure of the instruction execution.

DESCRIPTION:

The Generate New DEA Master Key instruction causes a 128 bit odd parity adjusted random number to be generated and stored in the new master key register (i.e., the NKM register). The left and right 64 bit parts of the generated key must be unequal. The instruction executes only if the NKM flag is in the "empty" state. Successful execution of the GNDMK instruction causes the NMK flag to be set to the "full" state from the "empty" state.

The GNDMK instruction executes only in the "run" state.

Set PKA Master Key (SPMK)

EQUATION:

```
                    ()
                    -->
                    CC                  /unspecified/
```

PARAMETER DEFINITIONS:

INPUTS	DESCRIPTION
	None.

OUTPUTS	DESCRIPTION
CC	Condition code indicating success or failure of the instruction execution.

DESCRIPTION:

The Set PKA Master Key instruction transfers the contents of the CKMP register into the OKMP register and then transfers the contents of the NKMP register into the CKMP register. This instruction operates only if the NKMP flag (new PKA master key flag) is in the "full" state and the left and right 64 bit parts of the key stored in the NKMP register are unequal. Also, if the CKMP flag is in the "full" state and CKMP HISTORY = 1, then the instruction operates only if NKMP HISTORY = 1. This guarantees that a CF-generated KMP can't be replaced by a CFAP-supplied KMP.

The SPMK instruction is used to activate a new KMP after the RTNPMK instruction has been used to reencipher encrypted records in the CKDS from encryption under the current KMP to a new KMP.

The SPMK instruction executes only in the "run" state.

Load MDC For Public Certification Key (LMDCC)

EQUATION:

```
                    index-value         /4b/
                    MDC-value           /128b/
                    -->
                    CC                  /unspecified/
```

PARAMETER DEFINITIONS:

INPUTS	DESCRIPTION
INDEX-VALUE	A 5 bit field containing an index value from 0 to 15.
MDC-VALUE	A 128 bit modification detection code to be loaded into one of 16 128-bit storage locations in MDC Table, designated as EKUMDC(0), ..., EKUMDC(15).

OUTPUTS	DESCRIPTION
CC	Condition code indicating success or failure of the instruction execution.

DESCRIPTION:

The Load MDC For Public Certification Key instruction permits a 128 bit MDC, designated MDC-value, to be loaded and stored in the CF in one of 16 possible storage locations in MDC Table, designated as EKUMDC(0), ..., EKUMDC(15). MDC-value is stored in EKUMDC(i), where i is the value of index- value. MDC-value contains an MDC calculated on an External Key Unit (EKU) using one of several possible hashing algorithms (see the hash-rule parameter of the IPUK instruction). The EKU must contain a public certification key PUC. (The fact that EKU contains a public certification key is verified when EKU is imported using the IPUK instruction.) The Load MDC For Public Certification Key instruction sets EKUMDC FLAG(i) equal to B'10'.

The LMDCC instruction operates only when EKUMDC FLAG(i) = B'00'. Other- wise, to load an MDC into an already occupied MDC Table location requires EKUMDC FLAG(i) to be reset to B'00'. This can be done only be issuing an EPS or EIS instruction. For reasons of security, the LMDCC instruction is architected such that the MDC Table locations EKUMDC(0) thru EKUMDC(15) cannot be changed without erasing the contents of the CKMP register. Thus, a certification center has the means to audit each security module to ensure that public certification keys have been loaded in conformance with an established network security policy.

The EKUMDC FLAG serves the following purposes: (a) it controls initialization of the MDC Table via the LMDCC and LMDC instructions, and (b) it controls import of public keys via the "MDC-mode" parameter of the IPUK instruction.

The ECFAR instruction can be used by CFAP to view the contents of the MDC Table and the EKUMDC FLAG field.

The LMDCC instruction executes only in the "run" state.

Load MDC (LMDC)

EQUATION:

```
MDC-value          /128b/
-->
CC                 /unspecified/
```

PARAMETER DEFINITIONS:

INPUTS	DESCRIPTION
MDC-VALUE	A 128 bit modification detection code to be loaded into EKUMDC(16).

OUTPUTS	DESCRIPTION
CC	Condition code indicating success or failure of the instruction execution.

DESCRIPTION:

The Load MDC instruction permits a 128 bit MDC to be loaded and stored in MDC Table storage location EKUMDC(16). MDC-value contains an MDC calculated on an External Key Unit (EKU) using one of several possible hashing algorithms (see the hash-rule parameter of the IPUK instruction). The EKU must contain a public key management key. a public authentication key, or a public user key (no public certification key). (The fact that EKU contains a public key management key, a public authentication key, or a public user key is verified when EKU is imported using the IPUK instruction.)

Unlike the LMDCC instruction, the LMDC instruction executes regardless of the current value of EKUMDC FLAG(16). Execution of the LMDC instruction causes MDC-value to be loaded into EKUMDC(16) and EKUMDC FLAG(16) to be set equal to B'10'.

The EKUMDC FLAG serves the following purposes: (a) it controls initialization of the MDC Table via the LMDCC and LMDC instructions, and (b) it controls import of public keys via the "MDC-mode" parameter of the IPUK instruction.

The ECFAR instruction can be used by CFAP to view the contents of the MDC Table and the EKUMDC FLAG field.

The LMDC instruction executes only in the "run" state.

Initialize Pseudorandom Number Generator (IPRNG)

EQUATION:

```
()
-->
CC                    /unspecified/
```

PARAMETER DEFINITIONS:

INPUTS	DESCRIPTION
	None.

OUTPUTS	DESCRIPTION

CC	Condition code indicating success or failure of the instruction execution.

DESCRIPTION:

The Initialize Pseudorandom Number Generator instruction initializes the pseudorandom number generator using the method specified in the Initialize Pseudo-Random Number algorithm (Initialize Pseudo-random Number). The Initialize Pseudo-random Number algorithm reads the current values stored in the PRNGKEY1, PRNGKEY2, and PRNGCTR1 registers and calculates two new key values which are then stored back into the PRNGKEY1 and PRNGKEY2 registers.

The IPRNG instruction executes in the "preinit", "init", and "run" states.

CF CONFIGURATION

Load Configuration Vector (LCV)

EQUATION:

```
config-vector      /512b/
-->
CC                 /unspecified/
```

PARAMETER DEFINITIONS:

INPUTS	DESCRIPTION
CONFIG-VECTOR	A configuration vector.

OUTPUTS	DESCRIPTION
CC	Condition code indicating success or failure of the instruction execution.

DESCRIPTION:

The Load Configuration Vector instruction permits a 64 byte configuration vector to be loaded and stored within the CF Environment. Execution of the LCV instruction causes the LCV FLAG to be set to the "full" state. The LCV instruction executes only when the LCV FLAG is in the "empty" state. The LCV FLAG can only be reset to the "empty" state via execution of an EPS or EIS instruction. In effect, the LCV FLAG controls LCV execution as follows: (a) If the LCV FLAG = "empty" state, then LCV instruction execution is enabled for one execution only, whereas (b) if the LCV FLAG = "full" state, then LCV instruction execution is disabled.

Execution of the EIS instruction causes a configuration vector in the CF Environment to be initialized/reinitialized to a "default" value. This value can be changed by executing an LCV instruction.

For reasons of security, the LCV instruction is architected such that the configuration vector value stored in the CF Environment cannot be changed without erasing or invalidating the contents of the CKMP buffer.

The LCV instruction executes only in the "init" state.

FUNCTIONAL SPECIFICATION:

1. Perform configuration vector and state vector checking:
   a. Verify that CF STATE in the state vector is in the "init" state.
   b. Verify that LCV FLAG in the state vector is in the empty" state.
   Continue if checking succeeds; otherwise set CC status flag and jump to step 5.
2. Perform consistency checking on config-vector:
   a. Verify Version Number = X'01'
   b. Verify KM RELOAD = B'0'.
   c. Verify DEFINE(EPS) = B'1' or DEFINE(EIS) = B'1' (i.e., either EPS or EIS or both are defined to prevent CF-reinitialization lockout)
   d. For i = 0 to 71, do the following: 1) Verify DEFINE(i) = B'1'. 2) Verify AUTH CONTROL(i) = B'0'.
   e. For i = 72 to start-inst-index minus 1, do the following:
      1) Verify DEFINE(i) = B'0'.
      2) Verify AUTH CONTROL(i) = B'0'.
   Continue if checking succeeds; otherwise set CC status flag and jump to step 5.
3. Load the value of config-vector into the configuration vector.
4. Perform state vector update:
   a. Set LCV FLAG to the "full" state.
   b. For i = start-inst-index to 143, process AUTH CONTROL as follows:
      1) If AUTH CONTROL(i) = B'0', then set AUTH(i) := B'0' and ENABLE(i) := B'00'
      2) If AUTH CONTROL(i) = B'1', then set AUTH(i) := B'1' and ENABLE(i) := B'11'
5. Produce output CC from CC status flags.

CONTROL VECTOR CHECKING:

None.

Export Crypto Facility Audit Record (ECFAR)

EQUATION:

```
process-mode        /2b minimum/
PUA-key             /1b minimum/
product-component   /1b minimum/
<hash-rule>         /3b minimum/     ; if process-mode = (1 or 2)
<IKU1-length>       /16b/            ; if process-mode = 2
<IKU1>              /unspecified/    ; if process-mode = 2
<RN>                /64b/            ; if process-mode = (1 or 2)
-->
cfar-length         /16b/
cfar                /unspecified/
<dsig1-length>      /16b/            ; if process-mode = (1 or 2)
<dsig1>             /unspecified/    ; if process-mode = (1 or 2)
CC                  /unspecified/
```

PARAMETER DEFINITIONS:

INPUTS	DESCRIPTION
PROCESS-MODE	The process-mode parameter specifies the type of processing to be performed: o process-mode = 0 : no digital signature is generated o process-mode = 1 : a digital signature

is generated on CFAR1 using the private authentication key PRA stored in the PRA buffer in the CF.

- process-mode = 2 : a digital signature is generated on CFAR1 using the private key PR specified in IKU1.
- process-mode = 3 : reserved

PUA-KEY The PUA-key parameter indicates whether the cfar should contain cfpkr1, which contains the PUA key:

- PUA-key=0 : no
- PUA-key=1 : yes

PRODUCT-COMPONENT

The product-component parameter indicates whether the cfar should contain the Nonsecret Product Environment:

- product-component=0 : no
- product-component=1 : yes

HASH-RULE Specifies the hash algorithm to be used to calculate a hash value on cfar. The encoding of the hash-rule is as follows:

- hash-rule = 0 : MDC-2 algorithm
- hash-rule = 1 : MDC-4 algorithm
- hash-rule = 2 : MD4 algorithm
- hash-rule = 3 : quadratic residue
- hash-rule = 4-7 : reserved

This parameter is required only when process-mode=1 or process-mode=2.

IKU1-LENGTH The length of IKU1 in bytes. This parameter is required only when process-mode=2.

IKU1 An Internal Key Unit containing a private key PR. This parameter is required only when process-mode=2. The value of EID in SCB1 must equal the value in the EID register. The values of Tstart and Texp in SCB1 must satisfy the relationship Tstart $\leq$ DT < Texp, where DT is the current date and time expressed in Coordinated Universal Time.

RN A CFAP-supplied time-variant parameter to be stored in CFAR1. This parameter is required only when process-mode=1 or process-mode=2.

<u>OUTPUTS</u> <u>DESCRIPTION</u>

CFAR-LENGTH The length of cfar in bytes.

CFAR A Crypto Facility Audit Record.

DSIG1-LENGTH The length of dsig1 in bits. This parameter is required only when process-mode=1 or process-mode=2.

DSIG1 A digital signature produced from a CF System Signature Record (CFSSR) and a private key PR, in accordance with section 6 of ISO DIS 9796. The CFSSR contains a 128-bit hash value calculated on cfar. This parameter is required only when process-mode=1 or process-mode=2.

CC Condition code indicating success or failure of the instruction execution.

DESCRIPTION:

The Export Crypto Facility Audit Record instruction constructs a Crypto Facility Audit Record (CFAR) and returns it to the CFAP. The CFAR contains (1) a copy of the nonsecret part of the CF Environment, a date and time (DT) supplied by the CF, and (3) foo process-mode = 1 and process-mode = 2, a CFAP-supplied time-variant value RN. RN can be a random number, sequence number, or time stamp, which may be used by a designated receiving device to ensure that a produced CFAR is current.

A process-mode parameter specifies to the instruction whether a digital signature is generated on the CFAR and, if so, then whether the private key is (1) PRA or (2) a PR supplied to the ECFAR instruction. A hash-rule parameter indicates to the ECFAR instruction the hash algorithm to be used in generating the digital signature.

Process-mode = 1 can only be executed when the GDAK FLAG is in the "full" state. Process-mode = 2 can only be executed when the CKMP FLAG is in the "full" state.

The Export Crypto Facility Audit Record instruction executes in the "preinit", "init", and "run" states.

CF CONTROL

Enter Preinit State (EPS)

EQUATION:

```
                  ( )
                  -->
                  CC                  /unspecified/
```

PARAMETER DEFINITIONS:

INPUTS	DESCRIPTION
	None.

OUTPUTS	DESCRIPTION
CC	Condition code indicating success or failure of the instruction execution.

DESCRIPTION:

The Enter Preinit State instruction resets the CF STATE to the "preinit" state; it resets the configuration and state vectors to zero; it resets the POS register to value X'0123456789ABCDEF0123456789ABCDEF'; and it executes algorithm Initialize Pseudo-random Number to (further) initialize the pseudorandom number generator. The Enter Preinit State instruction DOES NOT erase or zeroize the PRNGCTR1, PRNGCTR2, PRNGKEY1, and PRNGKEY2 registers, which are registers used by the Initialize Pseudo-random Number.

Enter Init State (EIS)

EQUATION:

```
                  ( )
                  -->
                  CC                  /unspecified/
```

PARAMETER DEFINITIONS:

INPUTS	DESCRIPTION
	None.

OUTPUTS	DESCRIPTION
CC	Condition code indicating success or failure of the instruction execution.

DESCRIPTION:

The Enter Init State instruction loads a "default" configuration vector into the CF environment and resets certain flags in the state vector to change the state of the CF and to clear certain registers and buffers. (See "Configuration Vector" on page 32 for a description of the default configuration vector.) More particularly, the Enter Init State instruction causes the flags controlling the old, current, and new KMP registers to be reset to the "empty" state, thereby causing these keys to be invalid. It causes EKUMDC FLAG field to be reset to zero, thereby invalidating any MDCs currently loaded in the MDC Table. It causes the LCV FLAG to be reset to the "empty" state, thereby enabling execution of the LCV instruction. It causes the CF STATE to be reset to the "init" state.

The Enter Init State instruction does not reset flags associated with the master key KM.

The EIS instruction can be executed in the "preinit", "init", and "run" states.

Enter Run State (ERS)

EQUATION:

```
            ( )
            -->
            CC                    /unspecified/
```

PARAMETER DEFINITIONS:

INPUTS	DESCRIPTION
	None.

OUTPUTS	DESCRIPTION
CC	Condition code indicating success or failure of the instruction execution.

DESCRIPTION:

The Enter Run State instruction causes the CF STATE flag to be set to the "run" state.
The ERS instruction executes only in the "init" state.

Clear New PKA Master Key Register (CLNPMK)

EQUATION:

```
                ()
                -->
                CC                      /unspecified/
```

PARAMETER DEFINITIONS:

INPUTS	DESCRIPTION
	None.

OUTPUTS	DESCRIPTION
CC	Condition code indicating success or failure of the instruction execution.

DESCRIPTION:

The Clear New PKA Master Key Register instruction causes the NKMP flag in the state vector to be reset to the "empty" state.
The Clear New PKA Master Key Register instruction executes only in the "run" state.

Clear Old PKA Master Key Register (CLOPMK)

EQUATION:

```
                ()
                -->
                CC                      /unspecified/
```

PARAMETER DEFINITIONS:

INPUTS	DESCRIPTION
	None.

OUTPUTS	DESCRIPTION
CC	Condition code indicating success or failure of the instruction execution.

DESCRIPTION:

The Clear Old PKA Master Key Register instruction causes the OKMP flag in the state vector to be reset to the "empty" state.

The Clear Old PKA Master Key Register instruction executes only in the "run" state.

Set Authorization Flag (SAF)

EQUATION:

```
inst-index          /16b/
-->
CC                  /unspecified/
```

PARAMETER DEFINITIONS:

INPUTS   DESCRIPTION

INST-INDEX   An instruction and instruction-mode index referencing AUTH(inst-index) in the AUTH field of the state vector. inst-index is a positive integer value between start-inst-index and 143, inclusive. See Configuration Table for a definition of start-inst-index and end inst-index. See also AUTH field in the state vector.

inst-index is value referencing the following PKCD instructions and instruction modes:

110 VADS
111 SRALM
112 IPRNG
113 EPS
114 ECFAR
115 EIS

116 SAF
117 LMDCC
118 LMDC
119 LFPMKP (input 0)
120 LFPMKP (input 1)

121 CPMKP (input 0)
122 CPMKP (input 1)
123 GNPMK
124 GNDMK
125 CLNPMK
126 CLOPMK

127 SPMK
128 GPUPR (mode 0/2)
129 GPUPR (mode 1)
130 EPUK
131 IPUK

132 IPRK (input 0)
133 IPRK (input 1)
134 RTNPMK
135 RTCPMK
136 GKSP
137 IDK

138 GADS
139 GDS
140 VDS
141 ECFER
142 ICFER
143 VIKU

OUTPUTS   DESCRIPTION

CC   Condition code indicating success or failure of the instruction execution.

DESCRIPTION:

The Set Authorization Flag instruction permits an AUTH flag asssociated with a particular instruction or instruction mode to be set to the "authorization required" state. Initially, the AUTH flag may be in the "authorization not required" or "authorization required" state.

AUTH flags are reset to the "authorization not required" state via execution of an EPS or EIS instruction.

The Set Authorization Flag instruction executes in the "init" and "run" states.

Set Enable Flag (SEF)

EQUATION:

```
    inst-index          /16b/
    flag-val            /2b minimum/
    <ctr>               /8b/                ; if inst-index='CPMKP input-mode=0'
                                              or inst-index='CPMKP input-mode=1'
                                              or inst-index='GPUPR mode=0/2'

                        <r>                 /16b/
                        <V>                 /unspecified/
                        -->
                        CC                  /unspecified/
```

PARAMETER DEFINITIONS:

INPUTS DESCRIPTION

INST-INDEX An instruction and instruction-mode index referencing ENABLE(inst-index) in the ENABLE field of the state vector. inst-index is a positive integer value between start-inst-index and 143, inclusive. See Configuration Table for a definition of start-inst-index and 143. See also ENABLE field in the state vector.

inst-index is value referencing the PKCD instructions and instruction modes, as follows:

110 VADS
111 SRALM
112 IPRNG
113 EPS
114 ECFAR
115 EIS
116 SAF
117 LMDCC
118 LMDC
119 LFPMKP (input 0)
120 LFPMKP (input 1)
121 CPMKP (input 0)
122 CPMKP (input 1)
123 GNPMK
124 GNDMK
125 CLNPMK
126 CLOPMK
127 SPMK
128 GPUPR (mode 0/2)
129 GPUPR (mode 1)
130 EPUK
131 IPUK
132 IPRK (input 0)
133 IPRK (input 1)
134 RTNPMK
135 RTCPMK
136 GKSP
137 IDK
138 GADS
139 GDS
140 VDS
141 ECFER
142 ICFER
143 VIKU

FLAG-VAL A parameter specifying the ENABLE(inst-index) value, as follows:

- o 0 : enabled for any number of executions.
- o 1 : enabled for 1 execution only.
- o 2 : enabled for n (n= 1 thru 255) executions, where n is specified in input parameter ctr.
- o 3 : not enabled

The permitted values of flag-val for each instruction and instruction mode are listed below (see also the ENABLE field in the state vector for a description of ENABLE(inst-index) values which are valid and invalid).

inst index		0	1	2	3	inst index		0	1	2	3
108	reserved					127	SPMK	y	y		y
109	reserved					128	GPUPR (mode 0/2)	y		y	y
110	VADS	y			y	129	GPUPR (mode 1)	y			y
111	SRALM	y			y	130	EPUK	y			y
112	IPRNG	y			y	131	IPUK	y			y
113	EPS	y			y	132	IPRK (input 0)	y			y
114	ECFAR	y			y	133	IPRK (input 1)	y			y
115	EIS	y			y	134	RTNPMK	y			y
116	SAF	y			y	135	RTCPMK	y			y
117	LMDCC	y	y		y	136	GKSP	y			y
118	LMDC	y	y		y	137	IDK	y			y
119	LFPMKP (input 0)	y	y		y	138	GADS	y			y
120	LFPMKP (input 1)	y	y		y	139	GDS	y			y
121	CPMKP (input 0)	y		y	y	140	VDS	y			y
122	CPMKP (input 1)	y		y	y	141	ECFER	y	y		y